IEEE MICRO
Chips, Systems, Software, and Applications
FEBRUARY 1990

The First Hot Chips Symposium

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Cover by Design & Direction

IEEE MICRO

Volume 10 Number 1 (ISSN 0272-1732) February 1990

Published by the IEEE Computer Society

Departments

Feature Articles

IEEE Computer Society
PO Box 3014
Los Alamitos, CA 90720-1264
(714) 821-8380

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $19 in addition to IEEE Computer Society or any other IEEE society member dues; $35 for members of other technical organizations. Back issues: $10 for members; $20 for nonmembers. This journal is also available in microfiche form.

Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices.

Copyright and reprint permissions: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US Copyright Law for private use of patrons those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress St., Salem, MA 01970. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to Permissions Editor, *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Copyright © 1990 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

Editorial: Unless otherwise stated, bylined articles and descriptions of products and services reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society. Send editorial correspondence to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. All submissions are subject to editing for style, clarity, and space considerations. *IEEE Micro* subscribes to the Computer Press Association's code of professional ethics.

IEEE Micro

Editor-in-Chief
Joe Hootman
*University of North Dakota**

Editorial Board
Dante Del Corso
Politecnico di Torino, Italy

John Crawford
Intel Corporation

Stephen A. Dyer
Kansas State University

K.-E. Grosspietsch
GMD, Germany

David K. Kahaner
National Bureau of Standards

Jay Kamdar
National Semiconductor Corporation

Hubert D. Kirrmann
Asea Brown Boveri Research Center

Richard Mateosian
Hitachi America, Ltd.

Marlin H. Mickle
University of Pittsburgh

Varish Panigrahi
Digital Equipment Corporation

Ken Sakamura
University of Tokyo

Michael Slater
MicroDesign Resources Inc.

Richard H. Stern

James H. Tracey
University of Texas at San Antonio

Philip Treleaven
University College London

Carl Warren
McDonnell-Douglas Space Systems

Yoichi Yano
NEC Corporation

Maurice Yunik
University of Manitoba

Magazine Advisory Committee
Sushil Jajodia (chair)
Jon T. Butler
B. Chandrasekaran
Manuel D'Abreu
Joe Hootman
Ted Lewis
H.T. Seaborn
Bruce D. Shriver
John Staudhammer

Publications Board
Sallie Sheppard (chair)
Victor Basili
P. Bruce Berra
J. Richard Burke
J. T. Cain
B. Chandrasekaran
David Choy
James Cross
Manuel D'Abreu
James J. Farrell III
Joe Hootman
Glen G. Langdon
Ted Lewis
Ming T. Liu
C.V. Ramamoorthy
Bruce D. Shriver
John Staudhammer
Harold Stone
Steven L. Tanimoto
Joseph Urban

Staff
Editor and Publisher
H.T. Seaborn

Assistant Publisher
Douglas Combs

Managing Editor
Marie English

Assistant Editor
Christine Miller

Assistant to the Publisher
Pat Paulsen

Art Director
Jay Simpson

Design/Production
Joseph Daigle

Membership/Circulation Manager
Christina Champion

Advertising Coordinators
Heidi Rex, Marian Tibayan

Reader Service
Marian Tibayan

* Submit six copies of all articles and special-issue proposals to Joe Hootman, EE Dept., University of North Dakota, PO Box 7165, Grand Forks, ND 58202; (701) 777-4331 Compmail+ j.hootman

From the Editor-in-Chief

A special issue

The guest editor of this issue, Bob Stewart, did a remarkable job of obtaining articles for *IEEE Micro*. (Bob was one of the founding fathers of *Micro* some 10 years ago.) He also did a terrific job of chairing the Computer Society's Hot Chips Symposium from which these articles come. With the help and enthusiasm of program co-chairs Jack Grimes and Dave Ditzel, Bob provided an important forum for developers of advanced chips to present their work to an interested public.

Glen Langdon, chair of the Technical Committee on Microprocessors and Microcomputers, not only supported the idea for a technical meeting of this type but also worked to bring it to reality. Some 480 people gathered at Stanford University last June for the meeting, which featured keynote speaker and ACM Turing award winner W. Kahan.

To the best of my knowledge, this is the first time that *Micro* has dedicated an issue to articles that result from a conference. This work has an interesting evolution. The presenters started with a verbal/slide approach at the conference, rather than a formal written address. Bob then selected the most interesting and important topics for article development. The authors have since had time to write their manuscripts, refine the information presented at the conference, and add material to help explain their ideas and hardware.

The first article in this issue describes the development of low-power Sparc ECL chip technology by Sun Microsystems and BIT. Next, Dave Johnson of Stardent Computer discusses the problems that chip architects create for compiler writers and makes cogent suggestions to remedy these problems.

Author John Crawford of Intel describes the i486 chip, and Tom Pennello of MetaWare writes about the problems of interfacing C routines for various CISCs and RISCs, including the i860. A revolutionary set of chips and a board from Sun for accelerating graphics processing appear in Curtis Priem's article. Mark Birman and coworkers at Weitek then present new CMOS floating-point units that execute a double-precision multiply operation in 150 nanoseconds at 40 MHz.

The final article in this issue describes the Motorola 68040 processor with on-chip MMU, FPU, and caches. The second part of this article will appear in the June issue.

I commend Bob and the authors on the exceptional quality of the articles they have produced. In fact, they presented so much quality work for publication that we had to run two issues related to the Hot Chips Symposium. We plan the next one in June.

In addition to the second installment of the 68040 article, June will present a discussion by Kahan on stack-overflow problems of the 8087 family floating-point unit. That issue will also contain an article on byte sequencing and arrangement of data on buses, one on a Texas Instruments floating-point unit for Sparc applications, and another one

In the mailbag

April 1989

"I would like to see more information about the latest developments in microprocessor technology." J.M.V., Lomas de Zamora, Argentina (You should enjoy this issue and its follow-up, which will present the latest in the development of new chips.—J.H.)

"I liked all the articles in this issue. I would like to get technical data on DSP chips and the 88000." V.R.M., Bangalore, India (Your request for information has been forwarded to Motorola.—J.H.)

"I liked the appearance and content of the magazine. I disliked the large number of acknowledgments. I would like to see more diagrams." N.P.H., Montchanin, DE (A diagram is worth 1,000 words, and we encourage authors to use drawings when appropriate. However, figures do take space away from the text, and with a fixed page budget we must compromise between text and figures at times.—J.H.)

June 1989

"I would like to see text-to-speech conversion—the latest trends." S.N.P., Bangalore, India

"I would like to see a complete list of neural network research centers in Europe." G.H.J.N., Nijmegen, Netherlands (I have forwarded your request to individuals in Europe who may be able to help you.—J.H.)

"I would like to see [an article on the] 68000 microprocessor." V.J.A., Mexico, D.F. Mexico

August 1989

"I would like be sent detailed information about the Intel i860 microprocessor." S.R.E.R., Yemen Arab Republic (I have sent your request to Intel.—J.H.)

"I would like to see articles on GaAs, DSP, RISC, vector processors, and VLSI architectures." V.P., Kuopio, Finland

"I liked New Products. [It] gives me information about the latest technology available on the market." H.S., Jakarta, Indonesia

"I liked your information about 64-bit processors. The information is not well written. I would like to see [it] in more detail." P.D.K., Kadi Mehsana, India (At least three reviewers examine each article submitted to *IEEE Micro*; two of these reviewers must agree that the article is worth publishing. Then the staff edits the manuscripts for clarity and house style to produce consistency of form throughout the issue. I believe that this process makes for a readable, concise, and informative magazine.—J.H.)

"I liked the clear, defined layout, giving details of authors, etc. I disliked the 'over-editing' of people's letters. I would like to see more space devoted to reader discussion (should the input be forthcoming)." D.K., Cambridge, England (We encourage reader input and read all material that is sent to us. We really like to have reader commentary. All material that is in good taste and has a bearing on the information that readers need to hear about appears in print. If letters are edited, they do not appear unless the author of the letter approves of the editing.—J.H.)

"I would like to see a more detailed description of some development software tools from a practical point of view—perhaps an annual book guide?" A.L., Waterloo, Belgium (One of the *Micro* Editorial Board members is getting together an article on engineering software tools. I am not sure when it will be published.—J.H.)

"I would like to see something on bus structures, and a comparison of Motorola and Intel chips." S.Y.O., Seoul, Korea

on the Motorola family of 88000 processors.

The second Hot Chips Symposium is planned for August of 1990. Hasan Alkhatib of Santa Clara University will act as general chair, with Alan Jay Smith of the University of California, Berkeley and John Crawford of Intel as program co-chairs. If you have a paper for consideration, contact them on electronic mail at smith@berkeley.edu or crawford@mipos2.intel.com.

Publishing two issues a year on one theme is not new to *Micro*. Each year the magazine devotes two issues to the international developments in the microcomputer industry. Because we devote about one third of our publishing efforts to international issues, we became interested in how many international readers we had. In a recent survey of these subscribers, Assistant Publisher Douglas Combs found that the number of *Micro* readers per country are

- Canada—1,151,
- Australia—558,
- Japan—365,
- Germany (Federal Republic and West Berlin)—267,
- England—255,
- Spain—243, and
- Switzerland—220.

In addition, 33 other countries have between 10 and 200 readers, and 49 countries between one and 10. The readership totalled 5,742—in some 90 countries! That figure equals about one third of our overall subscriber list. This international character of *Micro* certainly contributes to the uniqueness of the magazine.

Department

Micro World

Hubert Kirrmann
Asea Brown Boveri Research Center
CRBC.1
CH-5405 Baden, Switzerland

Cocom: The next wall to fall?

Are you aware that mailing a spare 8086 processor to a friend in East Germany could jeopardize national security? Look carefully at the supplier's bill. A small notice advises that reexport of this part is prohibited and that all responsibilities rest with you as the new owner. To mail it legally, you need a certificate from the government export office attesting that the part is not strategic ware. No sane person would think an 8086 belongs to that category, but the odds are high that it does. Since custom agents are seldom understanding, you cut the red tape by gluing a screw to the chip and mailing the whole thing as bolt and nut.

You can afford to cheat with the export rules as a private citizen, but not as a firm that trades with foreign countries. Dramatic consequences can result. But who makes these export rules?

Cocom (the Coordinating Committee on Multilateral Export Controls) is a voluntary trade organization based in Paris. NATO (North Atlantic Treaty Organization) formed the committee in the early Cold War period to prevent Communist countries from gaining access to other nations' high-level technology. Later on, Cuba, Iran, Libya, Nicaragua, the Philippines, and the Republic of South Africa joined the ban list. All NATO countries (except Iceland), France, and Japan are Cocom members. Neutral countries like Switzerland are free not to join. However, since they would have been treated like "impure" countries if they had broken the rules, they became de facto (and eager) members. Even cosignatories of the Cocom treaty can be blackballed. Cocom denied France Control Data computers in the 1960s because they were to be used in the modelling of its hydrogen bomb.

Cocom reaches decisions by unanimous agreement. In practice, one country can veto an item. For example, the US Bureau of Export Administration (mainly) works out lengthy and painstaking lists of strategic goods. These lists include all high-tech components and devices, and the systems that contain them. Strategic goods also include the tools and methods to manufacture the foregoing. Cocom can prohibit an export even if the parts do not originate in the US. Being manufactured under an American license qualifies them to be banned. Even seemingly harmless devices fall under the dual-use restriction. A sewing machine could be used for making uniforms, a PC could be used by the soviet intelligence agency, KGB, a mobile telephone by the Red Army.

Under the present rules, it is practically impossible to export a contemporaneous piece of electronics. Even under the update regulations of January 1990, you need a special permit to export

- dynamic RAMs or EPROMs (erasable programmable read-only memories) of more than 4 Kbytes,
- microprocessors having a word size greater than 8 bits or including a multiplier,
- floating-point units capable of adding 13-digit numbers in less than 20 milliseconds, and
- any IC with a clock rate of more

0272-1732/90/0200-0004$01:00 © 1990 IEEE

than 10 megahertz or that operates in the $-20C < T < 75C$ degree temperature range (the industrial range is $-25C < T < 85C$ degrees).

As a result, you cannot even export a knitting machine. The eight-year-old 68000 it contains still poses a military threat under the guidelines.

Progress in technique presents a challenge to those at Cocom who keep the catalog of strategic items up to date. As a consequence, the catalog enlarged considerably in the last decade. On the other hand, purging it has not been a priority task. Some items have been removed from the list, but their combination with each other is not allowed. For instance, CNC (computerized numerical control) cutting machines and CO_2 lasers are not presently restricted, but a CNC machine with a CO_2 laser is a strategic good. Conversely, a machine that contains a vetoed high-technology part may be exported if the value of the part is only a fraction of the total. Consequently, you may deliver a machine but not its spare parts.

Changes occur very slowly and depend on the political climate. For instance, the allowed transmission rate for multiplexers exported to the People's Republic of China has been increased from 8.5 to 45 Mbits/s. That was before the Tien-Amen Square massacre, which gave the hard-liners in Cocom new wings and stopped the negotiations between Cocom and the equipment suppliers. On the other hand, the Eastern countries are making progress. Recently, Cocom prohibited exportation of a machine because it contained high-quality bearings made in Poland.

Like all blockades, Cocom proved its inability to stop technology transfer to the East. The weapons of the Soviet Union are not lagging behind the West because of export restrictions. For experimental purposes, the soviet military always got what it wanted. The lack of hard currencies hampers scientists in Bulgaria or Poland more than export restrictions. The Soviets view massive use of Western components in their military equipment with the same suspicion as the US Army views massive use of Japanese ICs in theirs.

Specialized firms developed in the Far East, Austria, Switzerland, Canada, and the US to circumvent the rules. Their well-known methods include:

- carrying small items on trips (that's how universities can get their parts);
- selling materials to an embassy, which sends them on by diplomatic mail;
- taking equipment apart and sending pieces individually and legally;
- shipping equipment to a third country that reexports it;
- exhibiting equipment at a fair in the destination country and lending it out for "further examination";
- constructively hiding or renaming sensible components;
- sending software over telephone lines or by mail; and
- falsifying shipment papers of the equipment.

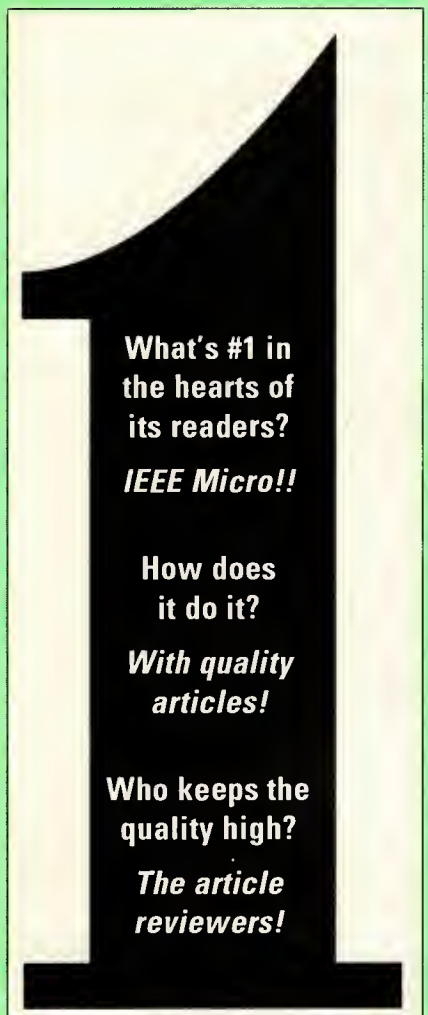
The smugglers' ingeniousness lengthens this list daily.

To fight technology smuggling, the US set up Operation Exodus in 1981. This Washington, D.C.-based Strategic Operation Division of the US Customs Bureau operates with 300 special agents and a budget of $8 million. In 1988, they uncovered 6,987 cases of smuggling worth $511 million. Their activity sometimes shows up in the press in reports like "secret agents prevent the Soviet Union from getting a supercomputer" (that was a 2-MIPS VAX 782 that few people would still buy in the West). However, the bureau complains that support from friendly countries is weak. But even when a country collaborates actively, it cannot prevent leaks. The recent participation of a West German firm in a combat gas factory in Libya proves it.

The most publicized success of the Bureau was the uncovering of illegal sales of NC (numerically controlled) machines to the Leningrad shipyards. These machines, manufactured by Toshiba of Japan and programmed by Konigsberg Vaopenfabrikk of Norway, allow the cutting of high-precision, quiet submarine propellers. As a result, the detection range of submarines dropped from 80 to 75 kilometers. The shipment documents attested it would be used for civilian works, but the exporters knew the true destination of the equipment. As a result, Cocom prohibited Toshiba from exporting to a Communist country for 12 months, causing losses of $37 million. The US market also vetoed Toshiba, causing additional estimated losses of $1.6 billion. The director of Konigsberg awaits judgment; the new director of Toshiba had to apologize publicly in the US press.

How do firms react? First of all, they panic at the idea of being involved in a Cocom case. A few years ago, former employees of a Swiss semiconductor firm ordered a manufacturing machine on their own and reexported it to an Eastern country. Although the authors left before the affair, the company was quarantined. Managers of well-known firms risk jail if they step foot in the US because of some obscure infringement (some of these people carry two passports). Doing business with sensitive countries requires carefully combing the exported hardware and removing all banned items. To obtain the license for an Iranian installation, a company had to replace a VAX with a PDP 11, which didn't make the engineers especially happy. In other cases, companies remove the floating-point coprocessors from a system to comply with rules and advise the client to obtain them by another channel.

Those who complain most about Cocom rules are not the Eastern countries. These complaints arise from the European industrial sector, particularly from West Germany. First of all, the red tape hampers trade and places absurd restrictions on exports. The feeling is growing that the restrictions serve strategic needs less than they keep discipline in the ranks. Many people viewed even the submarine propeller case as a power play against the Japanese industry rather than as a real threat. After all, the alleged reduction in detection range is small. Finally, the orientation of Cocom as an anticommunist organization prevented it from acting efficiently against third world countries that have since become greater threats. It couldn't prevent Pakistan, India, and



Become a reviewer and join the #1 team.

Want to help keep *IEEE Micro* #1? Editor-in-Chief Joe Hootman seeks more technical reviewers—people interested in seeing that the articles published in *IEEE Micro* continue to be of the highest quality. The technical review process is a crucial step in providing readers with correct and timely information so they can keep up with their ever-changing profession.

Send your professional information directly to:
Joe Hootman
University of North Dakota
PO Box 7165
Grand Forks, ND 58202

others from making their bombs, nor has it stopped Iran, Libya, or Iraq from building combat gas factories or long-range missiles.

The West Germans have been particularly upset recently because export licenses denied them allowed North American firms to get contracts. Concretely, this situation prevented Siemens AG of Munich from equipping eight Bulgarian towns with digital telephony. Cocom granted The Canadian Northern Telecom export licenses on the basis of its "obsolete technology." While West German firms were denied permits for an exhibition, the US firm, California Microelectronic Systems, exhibited the vetoed items as a Moscow representative of Hewlett-Packard, Logitech, Qualstar, Wyze, and other firms. This procedure already has some tradition. As ex-President Carter placed an embargo on the material for the trans-Siberian pipeline, he prevented West German firms from exporting the pumps, which were then supplied by US companies.

With the changes taking place in Eastern Europe, the Cocom rules are becoming a major stumbling block—for the Western countries. Europe considers the East as its next frontier. The rebuilding of these economies will primarily vitalize the economies of Western Europe, since most of the money made available to the Eastern countries will be spent in the West. The industry has no intention of rebuilding these infrastructures with obsolete materials. The reconstructed telephone system of Eastern Germany will be digital, and it will probably serve as a test field for ISDN.

Understandably, US companies view with suspicion the opening of the Eastern market. They claim *glasnost* and *perestroika* helped increase technology smuggling. Nobody denies it, but Europeans have more to gain from keeping these economies alive than from impeaching them. The Germans feel the sole fact that they speak the same language as their counterparts already worries the Americans. They also complain that Cocom is becoming a commercial instrument in the hand of the US to restrict their East trade. Some firms have already declared they will export illegally. The next showdown is in the wings. The planned free exchange of goods over the West German-East German border will make control illusory.

What is the outlook? The pressure of the European countries to lift Cocom restrictions for Eastern countries is growing. French Minister Beregovoy publicly stated in November that the rules need revision. British Conservative Member Atkinson presented a motion in the same month to the Western European Union (the nine kernel members of the European economic community, with Ireland, Denmark, and Greece on the waiting list). The motion asks for a radical change in the export rules. Even in the US, an industrial advisory lobby has formed in Washington, D.C. The Boeing Corporation has warned that the Eastern market could lose Airbus if the restrictions are not lifted promptly. Last year, US manufacturers obtained the removal of restrictions on IBM PC ATs. (This class of PCs includes 16-MHz 80286s and 68000s that have a memory capacity of up to 4 Mbytes and a hard disk storage of up to 140 Mbytes.) Siemens subsequently won a $1 billion contract with the USSR Ministry of Education for a PC AT compatible.

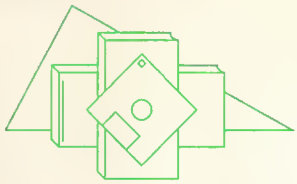
The US government does not seem to be thinking of giving up Cocom. On the contrary—the budget of Operation Exodus has been increased to $40 million and the personnel to 500. The question is whether anybody in Europe will care. The East raised the Wall to prevent free travel, the West raised Cocom to prevent free trade. Our Eastern friends tore down the Wall. Will Cocom be the next wall to fall?

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 174 **Medium** 175 **High** 176

Department



Micro Review

Richard Mateosian
Hitachi America, Ltd.
(415) 244-7456
Fax (415) 583-4207

The problems of Japanese-language computing

Many readers would like to know more about the Japanese language and the problems it presents for computing applications. This time I've reviewed a book that, despite its unlikely title, deals with exactly this problem.

The Fifth Generation Fallacy, J. Marshall Unger (Oxford University Press, New York, 1987, 240 pp., $24.95)

The title of this book derives from Unger's interesting but unproven theories about the political and cultural motivation for Japan's support of its Fifth Generation Project. (This project is a large-scale attempt to develop commercial applications of artificial intelligence, or AI.) More interesting to me are the author's detailed analysis and discussion of Japanese writing systems and the problems of using the standard writing system, kanji kanamajiribun, on computers.

Kanji kanamajiribun is the mixture of Chinese characters (called kanji in Japanese) with Japanese characters (called kana) from two essentially isomorphic syllabaries (called hiragana and katakana). Kana are like alphabets; letters of the roman alphabet can serve as standard representations of kana. Sections of Japanese represented in this way by roman letters are referred to as romaji. One can write Japanese entirely with kana, since kanji can be regarded simply as shorthand representations for specific sequences of kana. Hence one can write Japanese entirely in romaji.

The hiragana and katakana character sets comprise more than 100 symbols (depending upon which variations are regarded as distinct). Thousands of kanji exist. Knowledge of nearly 2,000 kanji is a minimum requirement to be considered literate in Japanese. Furthermore, no natural ordering—or easily remembered conventional ordering—exists for the kanji characters. There is no easy way to describe an unknown kanji or to find it in a dictionary.

For these reasons, dealing with the Japanese writing system of kanji kanamajiribun is difficult. Therefore, the fact that Japanese can be written entirely in romaji or kana provides a strong motivation for the elimination of kanji from computer-based applications. Counterbalancing this fact is the strong attachment of most Japanese people to their system of writing. They quite naturally dislike being forced to abandon a rich and subtle system simply to accommodate the inadequacies of a primitive technology. This is the essence of the conflict explored by Unger in his book. His thesis is that Japan's support of AI application development is based upon a belief that fifth-generation computers will allow them to keep kanji kanamajiribun as the standard writing system in computer applications.

The problems of storage and output of text represented by a large set of graphically complex characters are probably obvious to most *Micro* readers. As Unger points out, these problems lie mainly in the realm of hardware. Thus, while a cost differential between such systems and those using roman characters will always exist, the costs of both systems are expected to decline. Japanese users may well be willing to pay the differential.

The real problem, as Unger sees it, is efficient input. Unlike output, the real work of input, as he phrases it, occurs on the man side of the man/machine interface. In the absence of natural language recognition or mind reading—techniques Unger seems to consider to be equally likely to be developed—the user must convey the intended text to the machine through some input scheme.

0272-1732/90/0200-0007$1.00 © 1990 IEEE

Unger sets out four reasons why efficient input is the key issue:

- Practical problems easily handled by computers tend to be data intensive.
- A filter must process natural language data before it is presented to the machine.
- Effortless input forms the foundation for creative programming and documentation.
- Computer-supported, person-to-person sharing of ideas depends upon efficient input.

Unger analyzes a variety of schemes designed to support input of kanji characters. These schemes fall into three categories:

- inscriptive input, such as pen-point tracing or optical character recognition;
- transcriptive input, or strings of kana or romaji that are analyzed and converted to the intended kanji characters; and
- descriptive input in which every character has a code. The code is transmitted to the computer when a user either types the code explicitly or employs some variant of striking a key on which the character appears.

Transcriptive systems are the most popular in Japan because their relationship to spoken Japanese makes their use easy to learn. However, they require frequent interaction between user and machine because there is no unambiguous rendering of strings of kana into kanji kanamajiribun. The user must continually choose among alternatives proposed by the machine and modify incorrect renderings.

Descriptive systems, on the other hand, can be extremely easy to use, giving rise to Japanese touch-typing. However, no standard encoding encompasses all necessary characters—kanji, katakana, hiragana, romaji, other foreign character sets, and punctuation. Another drawback is that descriptive systems are not easy to learn to use. However, Unger cites a study by Hisao Yamada, a professor at the University of Tokyo, that sheds light on the difficulty of learning descriptive input. Patternless input is descriptive input in which there is no mnemonic relation between the key position and the character represented. Yamada's study shows that patternless input leads to faster mastery of touch-typing than schemes involving mnemonic arrangements of keys.

Unger is not reticent about what the Japanese should do. After 170 pages of analysis he concludes:

> Japan can remove all obstacles that prevent it from fully exploiting computer power by taking the following three steps: (1) separating those applications that absolutely require output in native script (e.g., typesetting) from those that do not; (2) adopting national standards for patternless descriptive input for the few applications found to fall into the former category; and (3) fostering public acceptance of romaji data processing in the majority of applications.

Seen as technical proposals, these conclusions constitute a reasonable approach to the problems of implementing Japanese-language computer applications with today's technology. The two issues that naturally arise are

- Would this approach discourage or retard the development of future technologies that might allow efficient use of kanji kanamajiribun in computer applications?
- Would the almost inevitable atrophy of kanji use cause a genuine loss to the Japanese people?

The first of these questions Unger dismisses by relegating such future technologies to the realm of science fiction. He doesn't believe that any currently visible program, particularly not AI, will lead to efficient kanji kanamajiribun use. His answer to the second question is that no loss would occur because the writing system is just a technological system distinct from the actual Japanese language. He regards the almost universal Japanese adherence to the opposite point of view to result from lifelong indoctrination with myths about kanji. Here he quotes Roy Andrew Miller's book *Japan's Modern Myth* but takes care to distance himself from Miller's "bitterly sarcastic terms" and his habit of "embroidering the facts for the sake of heightened rhetorical effect."

I can't go into detail about all of the myths and misconceptions Unger feels that the Japanese and others hold about kanji characters. The main one seems to be the "ideographic myth," a term apparently coined by John DeFrancis in his 1984 book *The Chinese Language: Fact and Fantasy*. This concept is well expressed in the following excerpts from Ezra Pound's *The ABC of Reading*, in which he explains Ernest Fenollosa's *Essay on the Chinese Written Character*:

> To go back to the beginning of history, you probably know that there is spoken language and written language, and that there are two kinds of written language, one based on sound and the other on sight. The Egyptians finally used abbreviated pictures to represent sounds, but the Chinese still use abbreviated pictures *as* pictures, that is to say, Chinese ideogram does not try to be the picture of a sound, but it is still the picture of a thing; of a thing in a given position or relation, or of a combination of things. It *means* the thing or the action or situation or quality germane to the several things that it pictures.
>
> Gaudier Brzeska, who was accustomed to looking at the real shape of things, could read a certain amount of Chinese writing without *any study*. He said, "Of course, you can *see* it's a horse" (or a wing or whatever).

Unger says that this ideographic myth prevents the Japanese from seeing that they are merely using kanji as "a burdensome collection of visual abbreviations."

Whatever you think of Unger's controversial opinions on political and cultural matters, the analysis and discussion of the practical problems of Japanese language data processing make this book worth reading.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 177 **Medium** 178 **High** 179

Micro News

Users said: No sacrifices!

Ware Myers
Contributing Editor

Outdoing the competition can be a taxing, yet rewarding, proposition. Consider the problems confronting a new firm when it tackled the design of a portable computer that would answer the needs of professional users. When Poqet Computer Corporation asked personal computer users what they would be willing to sacrifice for portability, they answered, "Nothing."

Up to now, vendors have had to compromise. If designers opted for the features users wanted, the computer turned out to be too heavy for users to move it about easily. If designers concentrated on reducing weight, they had to sacrifice features.

But Poqet Computer, based in Sunnyvale, California, and privately held between its employees and Fujitsu Corporation, wanted to give users the features they needed. Poqet also wanted to add low weight and small size to these features. Incredibly, Poqet's idea of low weight was one pound (0.45 kg), and its idea of small size turned out to be 8.75 × 4.3 × 0.925 inches (22.22 × 10.92 × 2.34 centimeters).

Its design team had to find new ways to accomplish some of the old functions.

IC memory replaces disks

A prime target of Poqet's design effort became mass storage. Heretofore rotating magnetic drives, either floppy disks or hard disks, had accomplished this function. But, a rotating drive's mechanical structure and electric motor are rather heavy. Also rotating the disk structure and moving the read-write head take a lot of energy. That, in turn, takes a larger power supply, and the power supply is usually the heaviest and largest component of a small computer. And, of course, supplying the power takes more batteries or more frequent changes of batteries.

Poqet's solution is solid-state, IC memory. Substituting it for rotating magnetic drives eliminated the major source of power drain in previous portables. The resulting computer, the Poqet PC, has no internal moving parts. It is powered by only two AA conventional alkaline batteries that last for up to 100 hours. With the intermittent use that a portable computer gets, the Poqet PC generally operates for several months between battery replacements.

The new computer employs both RAM and ROM. Users are expected to use RAM primarily to store the data that they themselves generate. If the data is to be maintained for any length of time, it can be transferred to a desktop PC or to the Poqet disk drive.

Designed as a companion to the Poqet PC, the $395 portable disk drive, using 3.5-inch, 1.44-Mbyte floppy disks, operates for 25 hours on four AA batteries at a 20-percent utilization rate. The drive is about the same size as the computer.

In addition to the internal RAM and ROM chips that come with the product, two slots accept optional RAM or ROM credit card-sized cards. Poqet expects the currently available 512-Kbyte RAM card to reach 2-Mbyte capacity in early 1990. The 640-Kbyte ROM card primarily supplies applications software. An important advantage these days, being read-only, it is virus-proof.

Programs and data in both types of memory are maintained when the computer is turned off. Both cards conform to a 68-pin physical format adopted by the Japanese Electronic Industry Design Association as a standard for IC memory devices. The recently formed Personal Computer Memory Card International Association is presently considering adopting this standard also.

Runs IBM PC software

The internal ROM supplied with the system holds MS-DOS version 3.3 and half-a-dozen built-in application programs. To make full use of the ROM concept, the system BIOS (basic input/output system)—itself located on the ROM—reads and executes applications software directly from ROM. BIOS loads only a minimal amount of the program into RAM. This technique frees a significant amount of the 512-Kbyte system RAM for working files. An added advantage: the ROM-stored soft-

continued on p. 94

FEATURE

Implementing Sparc in ECL

Two companies successfully joined forces to design a small, low-cost system capable of large mainframe performance.

Emil W. Brown
Anant Agrawal
Trevor Creary
Michael F. Klein
David Murata
Joseph Petolino

Sun Microsystems Inc.

In 1987, as the first Sun 4 workstation moved into full production, an emerging emitter coupled logic technology caught Sun Microsystems' attention. ECL promised very large scale integrated densities at much higher operating frequencies than CMOS (complementary metal-oxide semiconductor) technology. The convergence of this technology with our Scalable Processor Architecture (Sparc) would for the first time enable a complete microprocessor to be implemented in ECL.

Sun initiated a joint development project with Bipolar Integrated Technology to implement Sparc using BIT's new bipolar process. Initial work indicated that a clock cycle time of 12.5 nanoseconds, an execution rate of 1.3 clock cycles per instruction, and benchmark performance of 60 million instructions per second and 12 million floating-point operations per second were achievable with an entry-level price of $100,000.

This was an aggressive goal considering that mini-supercomputers then under development targeted clock rates of 25 to 40 megahertz and $200,000 to $500,000 entry-level prices. We were confident that the simplicity of Sparc would put the processor core on less than a dozen chips, and Sun's workstation heritage would fit the entire 80-MHz processor onto one circuit card. We minimized the cost by using air cooling, pin grid array and dual in-line packaging, 10K technology, and conventional printed circuit boards.

We believe that early adoption of new technology is the key to creating competitive products. However, chip development can no longer be separated from system design since much of the system now resides on the silicon itself. With these factors in mind, we assembled a design team that consisted of an equal number of IC engineers and system or board-level designers. We knew that board-level issues such as RAM access characteristics, transmission line design, and clock skew control as well as system architecture would determine many of the requirements for the VLSI chips.

Here, we briefly review both ECL technology and the features of BIT's ECL technique and discuss how board and cache considerations influenced the chip designs. Discussion of the integer unit pipeline, system interface signals, and coprocessor interface concludes the article. The chip set, now completed, sells commercially from BIT as the B5000 series. See The B5000 Microprocessor box.

Technology

ECL is a digital bipolar technology generally used for applications in which cost and power dissipation are less important than switching speed. The relatively large bipolar transistors of a traditional ECL design resulted in circuits with lower integration density than their CMOS counterparts. In addition, the power dissipation in ECL was high because the gates had to be biased to drive longer, more capacitive internal signal lines.

0272-1732/90/0200-0010$01.00 © 1990 IEEE

The B5000 Microprocessor

The B5000 Sparc integer unit is the first highly integrated ECL microprocessor (Figure A). Its design took a custom cell-based approach in which nearly 600 unique logic gates were employed. The data path, in the center of the die, performs all of the 32-bit-wide operations and contains the 32-bit-wide registers. Routing in this section is primarily vertical. The control section on the left side of the die uses cells of a fixed vertical pitch and is routed vertically on the third layer of metal and horizontally on the second layer of metal. Bias and supply lines appear in fixed vertical positions and run horizontally.

The register file, a 3-port RAM, contains 4,352 bits, uses 70,000 transistors, and provides a read access time of 10 nanoseconds. Two rows of pads on a 4.5-mil pitch provide contacts for 213 signal wires and 87 power bond wires. By designing custom input and output logic, we minimized setup and hold windows and output delays. The B5000 contains 122,000 transistors and 77,000 resistors in a 279-pin grid array. Some features of the chip include:

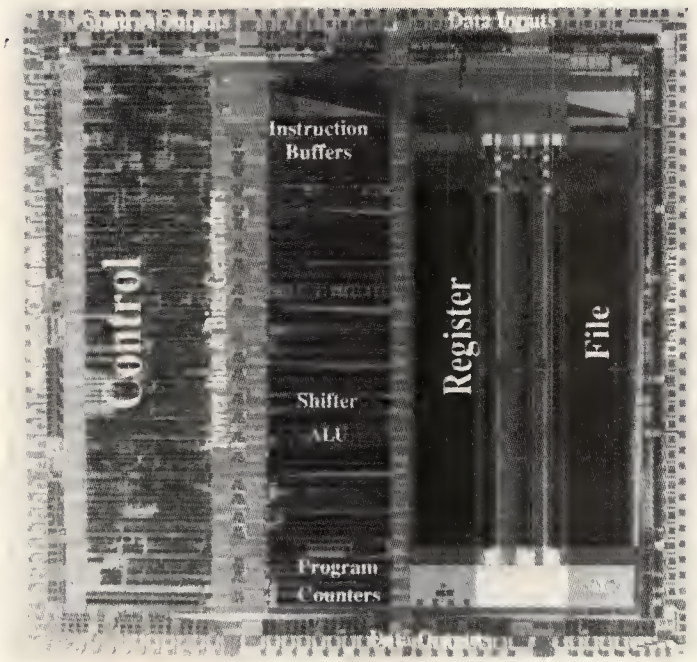


Figure A. A photomicrograph of the B5000 Sparc 375 × 387-mil microprocessor.

• **Parity**. Parity checking on incoming data occurs across each byte, and two parity bits protect each general-purpose register. Parity violations cause synchronous traps to support graceful error recovery and reporting. Data and register file parity checking can be independently disabled.

• **Scan mode**. Most of the registers in the chip set can be connected into one shift register by asserting the scan mode pin. The entire shift register can then be read and written by clocking the part. This feature is useful in testing the part, debugging a system containing it, and diagnosing failures.

• **Pass-through mode**. Two address pass-through modes allow the external cache controller to index each 64-bit word in a cache block, or to index the entire cache. The data pass-through mode supports write-backs and cache interrogation by allowing data from the cache to be shunted directly through the chip to the store-data bus.

• **Delay matching**. The 32 pins of the cache address bus, the two write enables, and the clock output are designed to switch within a 0.5-ns time window. The clock output can generate the system clock, which strobes the cache data into the IU or the FPU, and the write enables can be positioned precisely with respect to the addresses to simplify write-cycle timing.

• **Direct drive**. Address lines 3-18 use 25-ohm complementary output drivers to allow the chip to directly drive a 64-bit cache RAM array as large as 512 Kbytes. An on-chip register, which contains the address of the latest cache miss, further supports direct addressing.

• **Redundancy**. A redundant block of eight general-purpose registers can be substituted for any one of 14 blocks of window registers to provide yield enhancement. On the IC tester, a small set of test vectors determines whether a bad block of registers exists, performs any substitution, and indicates an encounter with one bad block. A simple firmware routine automatically performs the substitution in the system environment.

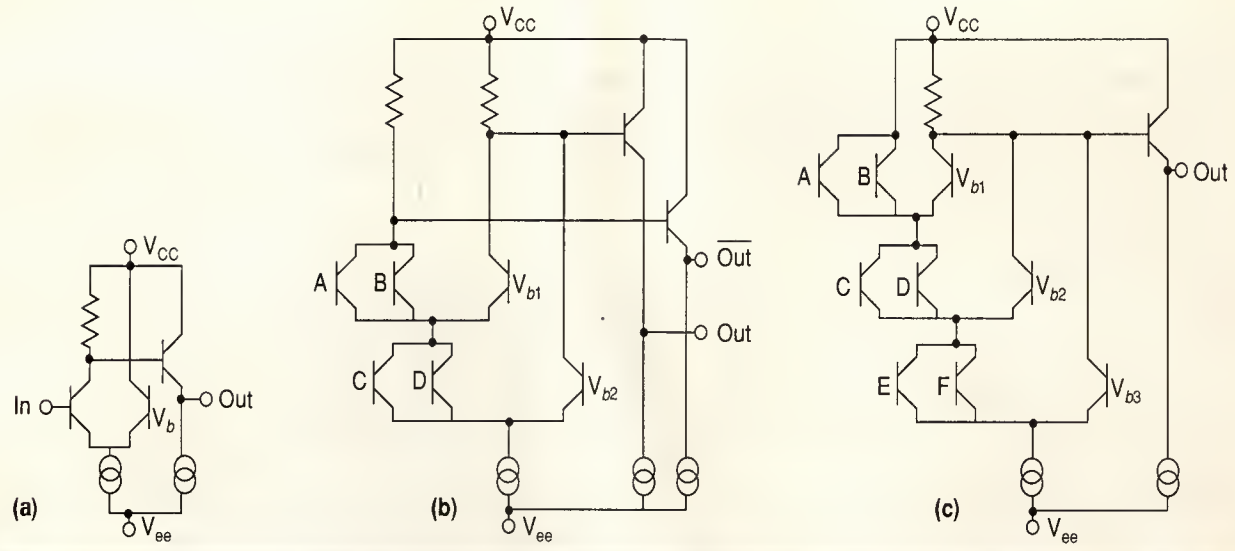


Figure 1. An ECL inverter (a), a two-level series gate implementing the function $(A + B)(C + D)$, and its complement (b), and a three-level series gate implementing the function $(A + B)(C + D)(E + F)$.

BIT achieved a double breakthrough when it

- reduced the physical size of the bipolar transistors, and
- provided a typical unloaded gate propagation delay of 375 picoseconds, while biasing each gate with 70 microamperes of static current. More traditional ECL techniques use 200 μA to 1 mA to achieve comparable switching speed.

With three layers of interconnect metallization, the process is ideally suited for building VLSI devices.[1,2]

All three layers of metal distribute power on these devices to minimize the voltage drops along the bus and to avoid metal migration. A package with an embedded copper-tungsten slug having high thermal conductivity dissipates the power. The die bonds directly to the slug, which transfers the heat to the top of the package. A heat sink in a forced airstream dissipates the heat. The resulting thermal resistance (θ_{ja}) is about 2.5 degrees centigrade per watt.

An ECL inverter consists of a differential pair, a current source, a load resistor, and an output driver (Figure 1a). By adding transistors in parallel to the input transistor of the differential pair, we create an Or function. If we add another differential pair between the original pair and the load resistor, we create an And function.

This stacking of differential pairs is called series-gating. A traditional ECL process allows only two levels of series-gating. For example, the logic function $(A + B)(C + D)$ can be implemented as one gate (Figure 1b). The BIT process allows three levels of series-gating, which supports functions such as $(A + B)(C + D)(E + F)$, as shown in Figure 1c. The penalty for the additional functionality is an increase in the propagation delay; however, this increase is generally less than if the function was decomposed into two gates. Three levels of series-gating proved useful in constructing the shift registers that make up the diagnostic scan chain in the integer unit, or IU, and the floating-point controller, or FPC. The required multiplexer and latch functions combine into one gate.

ECL circuits can efficiently drive transmission lines at high frequencies. So the IU and the FPC each contain a set of low-impedance (25-ohm) output drivers to drive system buses directly. We minimized the timing skew between these outputs by matching delays on the chip. We also designed package traces from the die to the pin to match the impedance of the drivers.

BIT technology supports two ECL interface standards, designated 10K and 100K. The 100K interface uses temperature compensation circuitry to minimize the temperature-induced shift in the switching threshold. The 10K interface specifies the amount of threshold shift that is allowed across the operating temperature range. All 100K circuits operate with a −4.5 volt supply, while 10K circuits operate at −5.2 volts. We chose to adhere to the 10K standard because it offers a wider selection of standard components. The cooling system maintains a maximum junction-temperature gradient of 25°C across the board to minimize differences in switching thresholds between devices. This system is necessary to maintain noise immunity and to control temperature-induced switching skew.

Board design influences

We started with several studies of cache designs. Although we looked at a variety of concepts, the simplest design offered the shortest access time, and the more complex designs did not offer comparable advantages.

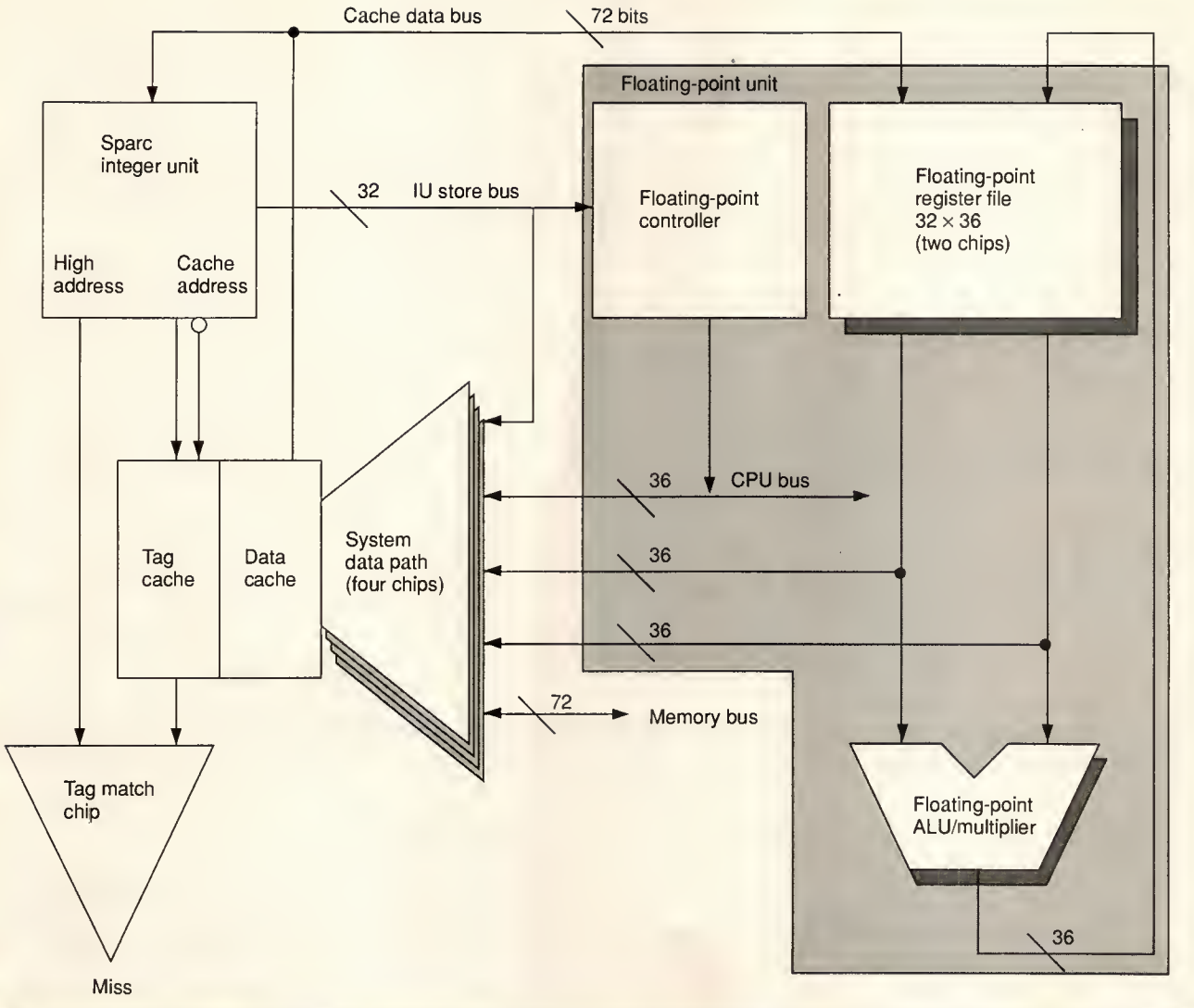


Figure 2. The CPU core.

We rejected set associativity because the set elements must be multiplexed, which inevitably increases the cycle time. We also discarded the idea of a nonpipelined, two-cycle access cache because branches become two-cycle instructions, reducing performance by 15 percent.

We strongly considered separate instruction and data caches, each with a 32-bit port. However, the lack of 64-bit data paths increases the cache miss penalty and adversely affects double-precision performance. Ultimately we chose a direct-mapped, write-back, 72-bit-wide (including parity), one-cycle access cache that contains instructions and data.

By fetching two instructions per cycle and using an instruction buffer on the IU, we achieve 80 to 90 percent of the performance advantage of separate caches. We designed the cache around a composite of several vendors' specifications for 16K × 4-bit and 4K × 4-bit RAMs. The write cycle takes longer than the read access time because address and data must be set up to the leading edge of the write pulse and held beyond its trailing edge. The write pulse itself is only slightly shorter than the access time.

The electrical and physical architectures are as important in high-speed ECL design as the logical architecture. Signal reflections and cross talk must be analyzed and minimized. In particular, we considered high-speed signal lines as transmission lines. They are driven from a single driver at one end of the wire. The wire must be daisy-chained from receiver to receiver and terminated at the other end by a DC impedance that matches the characteristic impedance of the line. To reduce cost, we did not specify differential transmission lines, except for lines carrying critical clock signals. We put power planes between signal planes and interleaved vertical and horizontal signal layers to manage cross talk on the printed circuit board.

Figure 2 illustrates the core of the processor and

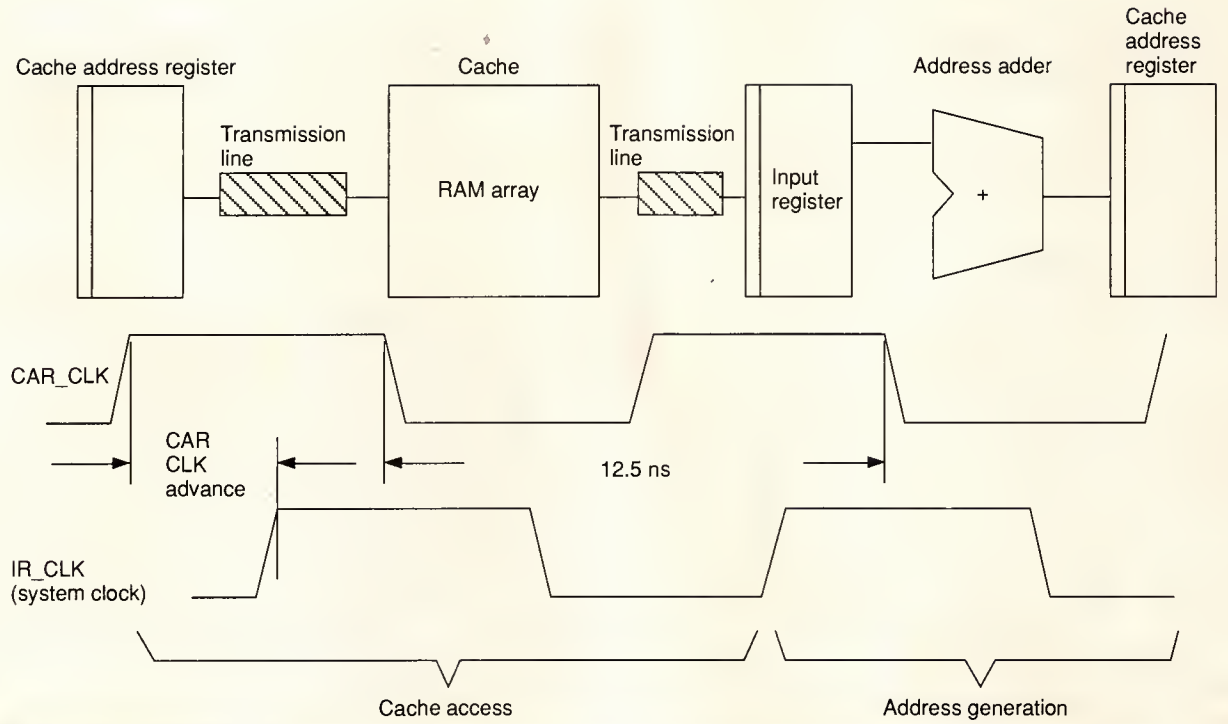


Figure 3. Address generation.

several basic decisions we made to minimize signal propagation delays:

- only the IU drives the cache address bus;
- the cache data bus only connects to the IU and the floating-point unit, or FPU; and
- the cache write operation does not limit the cycle time; it is preferable to use two cycles for each write.

As a result, all data entering the IU or FPU comes from the cache RAMs, and cache data must pass through the IU or the FPU to get to memory. Noncached data (such as system control registers, or power-on bootstrap code) moves through the cache. First, the cache controller saves the addressed word from the cache in a temporary register, writes the desired data into the vacated cache location, and from there into the IU or FPU, then restores the original cache data from the temporary register.

The CPU core seen in Figure 2 contains the Sparc integer unit, the five-chip FPU, the cache RAM array, the tag match chip, and four copies of the system data path gate array. The FPU consists of the controller, two register file chips, the double-precision ALU, and the double-precision multiplier. The tag match chip implements the cache miss logic and the tag portion of a four-entry translation (lookaside buffer) cache, or TLB. The system data path contains the data portion of the TLB and provides interconnections between the major units of the CPU via the CPU bus. It also contains the memory bus interface.

Cache design

With each address pin of a RAM presenting 5 to 7 picofarads of loading in DIP form, the AC impedance of an address bus with the RAMs packed as closely as possible is about 25 ohms. With 25 RAMs (18 for data and parity, and 7 for tags), the delay on the address transmission line totals about 10 ns. By splitting the cache into two banks, each driven by its own copy of the address, we halve the delay time to 5 ns. We provided the IU with 25-ohm differential drivers for 16 address lines, enough to support a 512-Kbyte cache. A further split into four banks is possible by using external drivers, but these can add significant delay and skew.

We optimized the tag-access and compare operations for speed in several ways. First, the tag RAMs are four times smaller than the data RAMs and thus have a shorter access time. Second, we designed a high-speed ECL gate array (the tag match chip) to perform the tag match computation, and third, placed the tag RAMs so that they are the first to receive the address from the IU.

Systems with one-cycle access caches have in the past required RAMs with a read-access delay significantly shorter than the cycle time of the machine. We assumed that such RAMs would be unavailable, too expensive, or too small for this machine, so we designed the pipeline to allow extra time in the cache access stage. We "borrow" time from the address generation stage (see Figure 3), which operates in less than 7.5 ns. An on-chip cache address register (CAR) is clocked with an early clock to enable the next cache

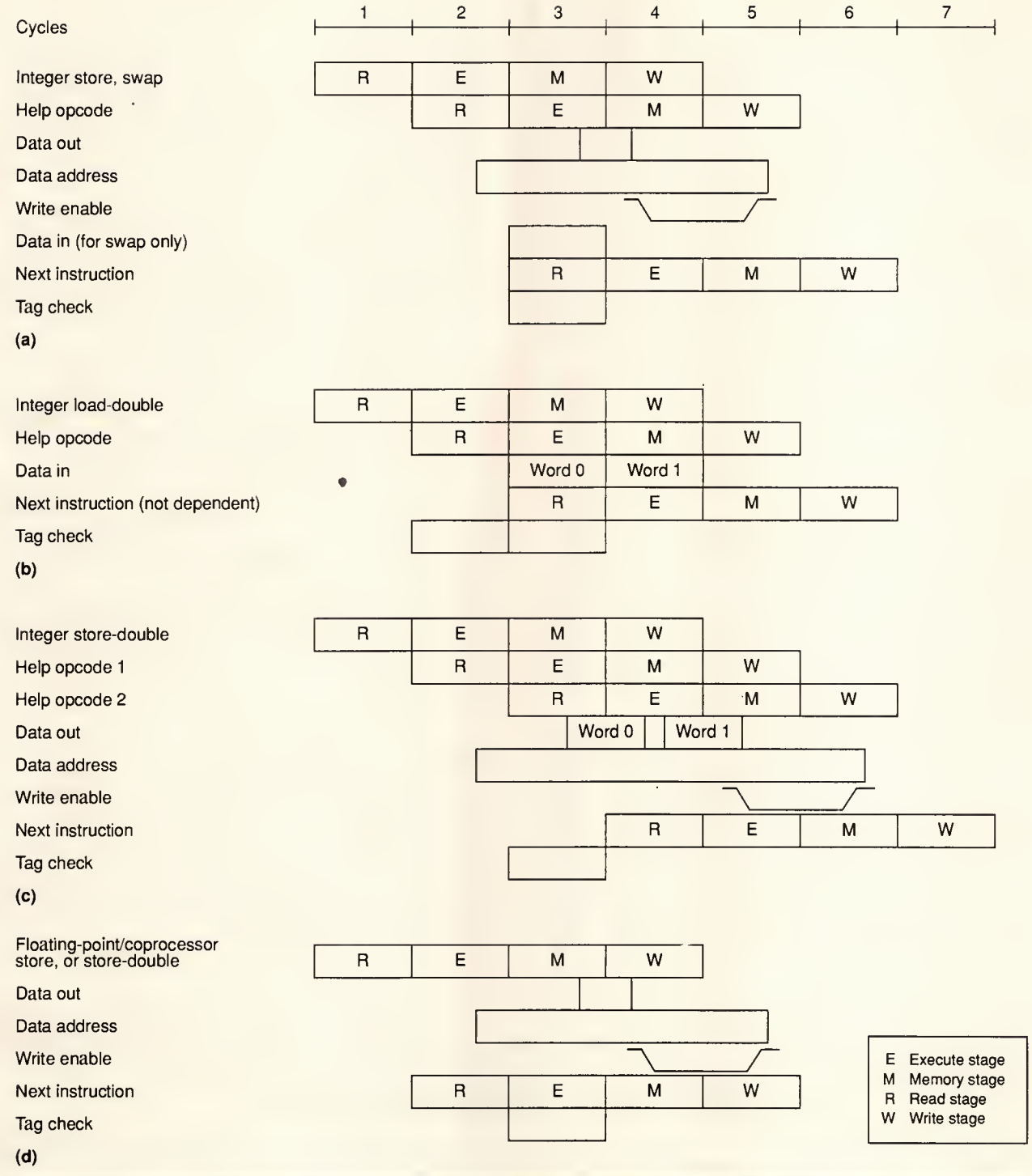


Figure 4. Various memory access instructions require multiple cycles on the B5000: store and swap timing (a), integer load-double (b), integer store-double (c), and floating-point store and store-double (d).

access to begin before the current one completes. We rely on the address transmission line and the RAMs to provide hold time for the data, despite the changing address.

As in previous Sparc implementations the store operation consists of a tag check followed by the actual cache write (on a cache hit).[3,4] In this implementation, the tag check takes one cycle, but the write takes two (see Figure 4). The write pulse straddles the second and third cycles because it must end before the early ad-

dress changes, and the store data arrives one cycle later than the address. There must also be time to stop the write pulse from occurring if the tag check results in a cache miss. If the store instruction is followed by a nonmemory-access instruction, its execution will be overlapped with the third cycle of the store, reducing the effective cost of the store to two cycles.

The timing of store and swap (seen in Figure 4a) illustrates the critical placement of the write pulse between the arrival of the data and the end of the address. Integer load-double (Figure 4b) takes two separate load cycles. Integer store-double (Figure 4c) is the only three-cycle instruction. Floating-point store and store-double instructions take one cycle in the pipeline, although it takes three cycles to complete the cache write (Figure 4d).

IU pipeline

Sparc offers a simple instruction set based on a register-to-register paradigm. Only load and store instructions access memory and the only addressing modes used by these instructions are register-plus-register and register-plus-immediate. Register specifiers appear in the same bit fields of every instruction. Branch instructions carry an immediate, program counter-relative offset.[5]

The simplicity of Sparc allows all of the instructions to fit neatly into a fixed pipeline which, on the B5000, consists of five stages: fetch (F), read (R), execute (E), memory (M), and write (W). (Note in The Sparc Architecture box that pipelining is not a required feature of any Sparc implementation.) Each stage completes its processing in one clock cycle. During the F stage instructions move from memory into the processor. Then the processor reads operands from the register file, decodes opcodes, and detects instruction dependencies in the R stage. The operands enter either the arithmetic and logic unit (ALU) or the shift unit in the E stage. Load and store instructions in the M stage fetch data operands from memory, and arithmetic instructions use it to move the arithmetic result back across the chip to the write port of the register file. In the W stage the processor writes the ALU result or the data from memory into the register file.

In the next clock cycle we use standard result-forwarding techniques to keep the pipeline full, even when an instruction's result is used.[6] We added a new data path to this implementation to allow load instructions to execute in one cycle, or two cycles when the next instruction requires the data being loaded. The hardware detects and handles the two-cycle case; compiler support is not required. We call this an interlock action and in general use it when an instruction encounters a resource conflict or data dependency and cannot be issued in the current cycle.

Figure 4 illustrates most of the multiple-cycle instructions. The integer store and store-double instructions require two and three cycles respectively because the IU register file does not have a third read port to access the stored data in the R stage. However, the floating-point load, load-double, store, and store-double instructions take just one cycle in the pipeline due to external 64-bit buses and the separate floating-point register file.

Instruction queue

The B5000 contains a 64-bit data input bus on which two instructions can be fetched in parallel. One or both of these instructions will be inserted into a queue, which has a maximum depth of four instructions. The queue allows the pipeline to complete one instruction every cycle even when memory access instructions occasionally use the data bus (Figure 5). The B stage of

continued on p. 19

The Sparc Architecture

The reduced instruction-set computer, or RISC, architectures developed at the University of California at Berkeley form the basis of the Scalable Processor Architecture. Sparc contains only 32-bit-long instructions in three formats. Operand specifiers appear in fixed positions in the instructions to enable rapid register file access. Delayed control transfer instructions allow an instruction that follows a control transfer instruction to execute before the transfer of control occurs.

In Figure B we can see that Format 1 is used only by the subroutine Call instruction. This format has a 30-bit displacement, which allows a Call to any word-aligned address in the virtual address space. Format 2 supports the Sethi instruction and the Branch instructions. The 22-bit displacement allows Branch instructions to span 16 Mbytes of the virtual address space. Format 3 supports all of the other instructions. It has three 5-bit operand specifiers, or two 5-bit operand specifiers and one 13-bit signed immediate constant. The three-

Format 1 (CALL instruction):

op	displacement
2	30

Format 2 (SETHI instruction):

op	rd	op2	immediate
2	5	3	22

Format 2 (Branch instructions):

op	a	cc	op2	displacement
2	1	4	3	22

Format 3 (Remaining instructions, i=0):

op	rd	op3	rs1	i	asi or fp-op	rs2
2	5	6	5	1	8	5

Format 3 (Remaining instructions, i=1):

op	rd	op3	rs1	i	immediate
2	5	6	5	1	13

a Annul bit, which conditionally inhibits execution of the branch's delay instruction
cc Condition code specifier
rd Destination register specifier
rs1, rs2 Source register specifiers

Figure B. Sparc instructions: 32 bits long in three formats.

operand version leaves 8 bits available for floating-point opcodes (fp-op), coprocessor opcodes (op), or alternate space identifiers (asi).

Integer computation occurs on only one data type, a 32-bit word, and always acts on operands from the register file. Only explicit memory load and store instructions access memory. Sparc supports word (32-bit), halfword (16-bit), and byte (8-bit) memory operations. All addresses must be aligned.

A windowed register file with two to 32 windows forms part of the Sparc architecture. Each window provides working storage for one subroutine of a program. Eight registers shared between adjacent windows support efficient passing of subroutine arguments. The save instruction, used at the beginning of a subroutine, advances to the next window while simultaneously performing an addition (usually to advance the stack pointer). The restore instruction permits return to the previous window when returning from a subroutine.

A separate set of 32 registers supports floating-point operations. All floating-point operations that refer to register operands implicitly access this set, including the floating-point memory load and store instructions. By their nature, floating-point arithmetic operations take longer than integer operations (using the same implementation), so Sparc allows floating-point operations to run for several cycles in parallel with integer operations. The separation of integer and floating-point data into two distinct register files eliminates direct dependencies between integer and floating-point instructions.

Table A.
Three of the four general types of the Sparc instruction set.

Instruction	Data types	Data width	Variations	Notes
Memory operations				
Load	Signed	Byte, halfword	Alternate space[1]	—
Store	Unsigned	Word	Floating-point queue	—
		Double word	Floating-point status reg.	—
		Single, double	—	—
Swap		Word	Alternate space[1]	Atomic[2]
Load/Store	Unsigned	Byte	Alternate space[1]	Atomic[2]
Integer computational				
And, Or	—	Word	Not[3]	—
XOR	—	—	Set cc[4]	
Add	—	Word	Set cc,[4] tagged[5]	—
Sub	—	—	Extended	—
MULSCC	Signed	Word	—	Set cc[4]
Save	—	Word	—	Performs Add
Restore	—	Word	—	—
Shift	—	Word	Left, right	Shift by 0 to 31 bits
Read	—	Word	PSR,[6] WIM[7]	—
Write	—	—	TBR,[8] Y[9]	—
Control transfer				
Branch	Signed	22-bit displacement	Integer cc[4]	—
—	—	—	Floating-point cc[4]	—
—	—	—	Coprocessor cc[4]	—
—	—	—	Execute delay[10]	—
—	—	—	Annul delay[10] if not taken	—
Call	Signed	30-bit displacement	—	Delayed
Jump and Link	—	Word	—	Delayed
Return from Trap	—	Word	—	Privileged, delayed

[1]Access to one of 255 privileged, alternate address spaces
[2]Load and Store occurs indivisibly in memory.
[3]The second operand is logically inverted.
[4]Condition codes
[5]The least significant two bits of the data act as simple type tags.
[6]Processor status register
[7]Window invalid mask
[8]Trap base register
[9]Y register (used by MULSCC as an accumulator)
[10]The instruction immediately following a control transfer instruction

Byte	8 bits
Halfword	16 bits
Word	32 bits
Double word	64 bits
Single	32-bit floating-point value
Double	64-bit floating-point value

In Tables A and B we list each individual instruction and its variations. Note that some implied variations don't exist. For example, Sparc contains no instructions for signed store, alternate space floating-point load or store, shift left arithmetic, or floating-point convert to self.

While pipelining is not a part of Sparc, the instruction set design allows efficient pipelined implementations. By keeping the instructions simple, it is relatively easy to overlap the execution of several instructions at once.

Table B.
The fourth general type of the Sparc instruction set.

Instruction	Data types	
Conversion	Signed	Integer
FADD, FSUB	—	Single
FMUL, FDIV	—	Double
FCOMPARE	—	—
Square root	—	—
FMOVE	—	Single
FABSOLUTE	—	—
FNEGATE	—	—

the pipeline represents cycles in which an instruction stays in the queue. By cycle 4, the chip holds three buffered instructions. When a load instruction uses the data bus to access data (Figure 5's shaded M stage in cycle 4), the pipeline uses up one of the prefetched instructions to maintain full performance, and the queue depth drops to two instructions in cycle 5.

The maximum depth of the queue determines how much load and store traffic can be tolerated without reducing the rate of instruction completion. With four buffers the machine can execute three back-to-back load instructions with no degradation. Since branches occur as frequently as every six cycles in typical programs, and cause the instruction queue to be flushed, we gain little performance increase by having a maximum queue depth greater than six or eight instructions. We chose a depth of four based on the chip area budget and performance simulations.

Conditional branching

The instruction following the Sparc branch instruction executes before the control transfer occurs. This delay allows the machine to execute a useful instruction while it adds the address of the branch instruction to its immediate offset to form the target address. On the B5000, the cache address for a given cycle issues late in the *previous* cycle. Since the generation of this address is a critical path, the B5000 predicts that every branch will be taken. It sends out the target address late in the R stage, without regard to the condition codes that are being computed in the same cycle by the compare instruction (Figure 6 on the next page, cycle 2).

If the branch is *not* taken, the "fall-through" instruction (the instruction sequentially following the delay instruction) is the next instruction to execute. In most cases this instruction resides in the instruction buffers, and no penalty results. In the case illustrated, we show the branch itself being fetched as the target of a previous branch. Thus it enters the R stage directly after it is

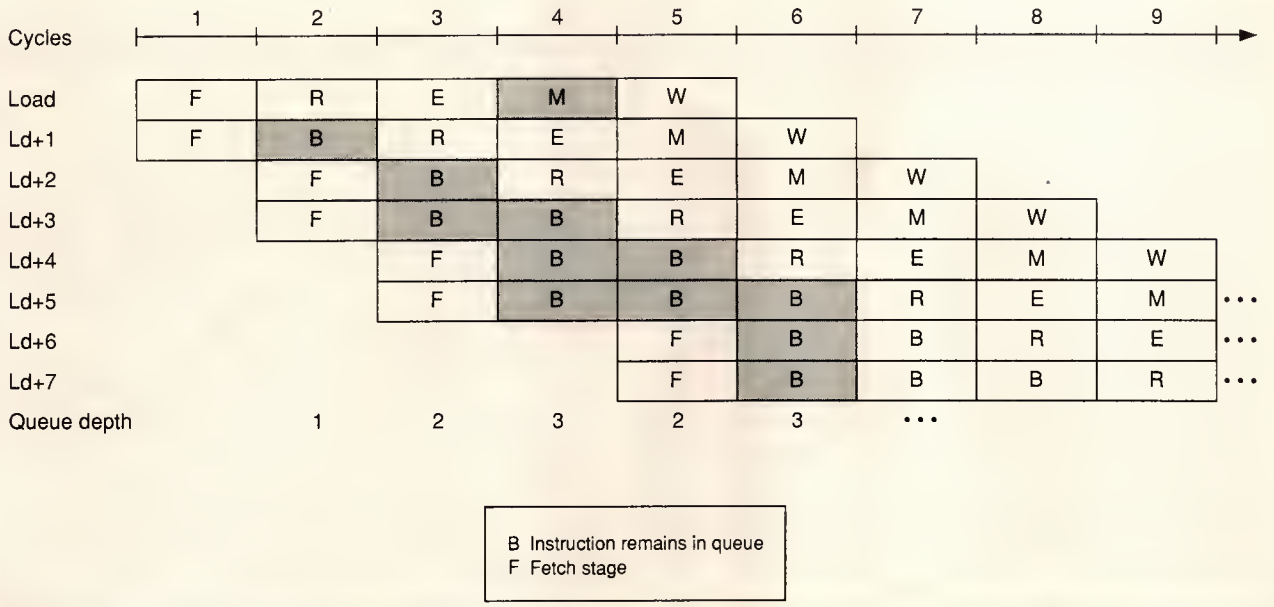


Figure 5. By fetching two instructions in each fetch cycle, the on-chip instruction queue fills as linear code executes.

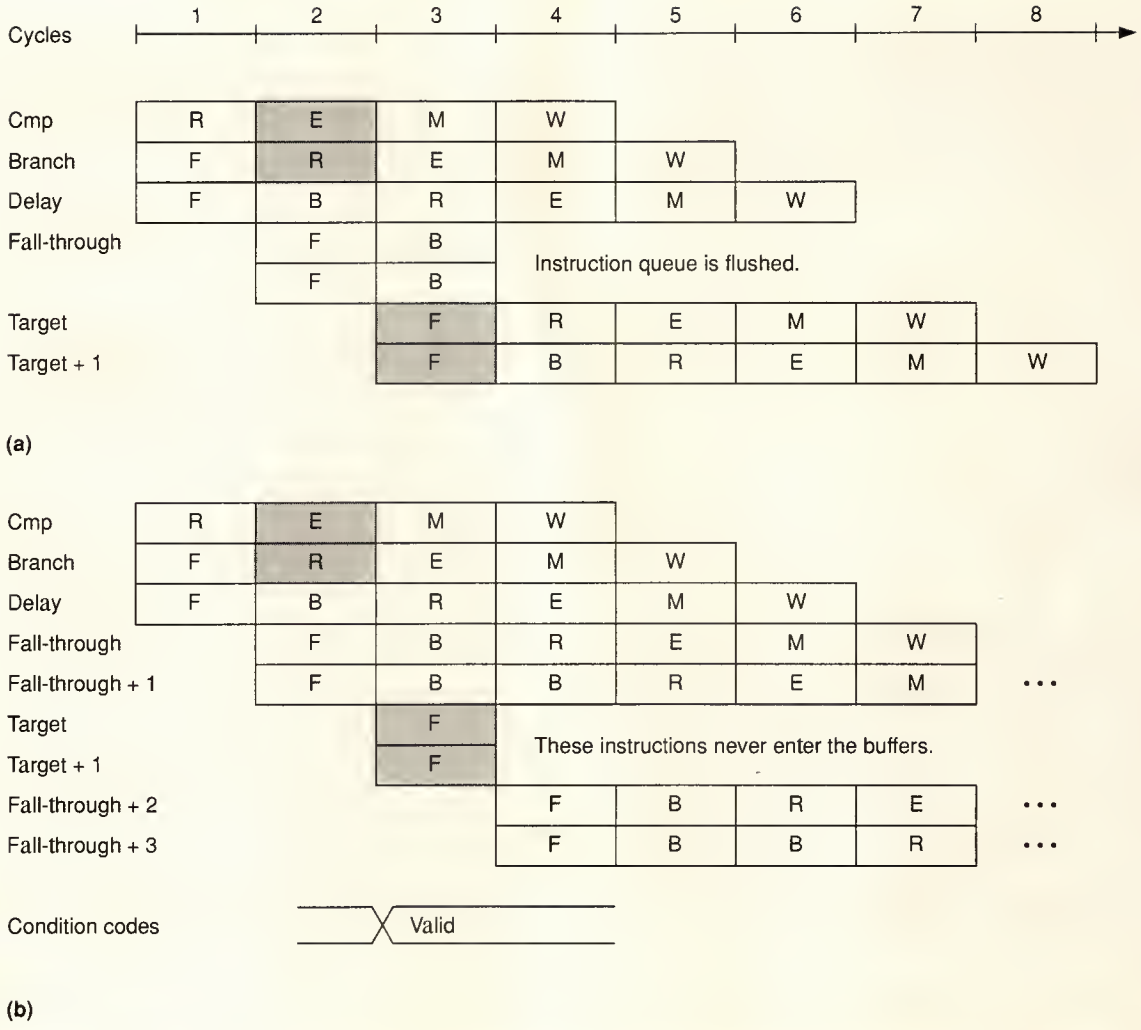


Figure 6. Branches: taken (a) and not taken (b).

fetched. The branch resided at an even-word address, so its delay instruction was also fetched at the same time. Complexities arise when the fall-through or delay instructions are not in the instruction queue. These infrequent cases do not significantly affect performance, although they do complicate the logic design.

The instruction at the target address of a branch is always fetched in the E stage of the branch (Figure 6). In the subsequent cycle 3, when the compare (Cmp) instruction computes and makes known the condition codes, the target is either decoded immediately or discarded. If the branch is not taken, the prefetched instructions following the branch/delay pair (the fall-through instructions) execute, so no cycles are lost.

FPU, coprocessor interfaces

The B5000 implements the Sparc floating-point interface and coprocessor interface symmetrically, so discussion of the floating-point interface applies equally to the coprocessor. Sparc allows register-to-register floating-point operations to execute and complete in the "background" while the pipeline continues to execute integer instructions. However, a floating-point operation may complete by generating an exception, rather than an arithmetic result.

To pinpoint the instruction causing an exception, the FPC maintains a queue of pending instructions and their addresses. The queue can be read after an exception occurs to determine which instruction actually caused the exception and which subsequent instructions had been issued but not completed. Each entry of the queue contains an instruction and its address. The IU dispatches instructions on the store-data bus in the R stage of the pipeline. The addresses are generated on the FPC chip, which has a copy of the E stage program counter (called the XPC). The IU manages the XPC with its "increment XPC" control, and by loading it from the store-data bus following any control transfer.

This design allows floating-point instructions to be dispatched at a rate of one every clock cycle.

When a dependency occurs between a floating-point arithmetic instruction and a floating-point memory access instruction, the FPC halts the IU until the arithmetic instruction completes. This process occurs in the W stage of the memory access instruction.

Three dedicated pins continuously carry the FPU's condition codes and a validation bit to the IU. The floating-point conditional branch instruction can thus execute in the pipeline as an integer instruction; it is not dispatched to the FPU. When a floating-point compare instruction is sent to the FPU, the unit invalidates the floating-point condition codes until the instruction completes. A subsequent conditional branch instruction must wait until the unit validates the condition codes.

Performance

The Dhrystone benchmark attempts to synthesize the frequency of occurrence of various high-level-language statements in operating system code. This benchmark is not a large program and doesn't really compute anything useful, but it has been run on many machines, from personal computers to mainframes. As a result, we find it useful as a one-dimensional measure of processor and compiler performance.

Reportedly, the Apple IIe with an 8-MHz 8086 processor executes 197 Dhrystones per second (version 1.1), and the IBM 3090/200 processes 31,250.[7] Simulation of the benchmark (version 1.1) on the B5000 indicates a rating of 104,000 Dhrystones/s, which is more than five times the rating of the Sun 4/260 workstation. Actual hardware results produced a rating of 103,000 Dhrystones/s. The difference occurs from process switching and the resulting cache misses.

From the Dhrystone simulation, we know that the B5000 should execute at the rate of 1.29 cycles per instruction and 62 million instructions per second. This is the expected average for integer benchmarks. Simulation of DAXPY (double-precision $A \times X + Y$) the double-precision inner-loop routine of the Linpack benchmark, on the B5000 has indicated a performance of 14 Mflops.

The B5000 ECL Sparc series of components represents a new approach to constructing high-performance computer systems at a relatively low cost. For the first time, we are able to design small ECL systems that perform at a level equal to large mainframe computers and approach that of today's supercomputers. In the future we can expect even more highly integrated bipolar and bipolar/MOS chips, further narrowing the gap between low-cost workstations and high-performance servers.

Acknowledgments

The B5000 series resulted from a productive joint development between Sun and BIT. In addition to the authors, many people contributed to the project, including Daniel Chang, Arthur Leung, Dan Nelsen, Renu Raman, Jim Testa, Kwok Tsang, and Alex Yu at Sun, and the layout and design teams at BIT.

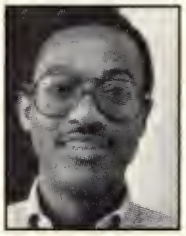
References

1. G. Wilson, "Creating Low-Power Bipolar ECL at VLSI Densities," *VLSI System Design*, May 1986.
2. "Electronics, Technology to Watch: Surprise! ECL Runs on Only Microwatts," *Electronics*, Vol. 59, Apr. 7, 1986.
3. M. Namjoo et al., "CMOS Gate Array Implementation of the SPARC Architecture," *Proc. Compcon 88*, IEEE Computer Society Press, Los Alamitos, Calif., 1988, p. 10.
4. M. Namjoo et al., "CMOS Custom Implementation of the SPARC Architecture," *Proc. Compcon 88*, IEEE CS Press, 1988, p. 18.
5. R.B. Garner et al., "The Scalable Processor Architecture (SPARC)," *Proc. Compcon 88*, IEEE CS Press, 1988, p. 278.
6. D.A. Patterson, "Reduced Instruction Set Computers," *Comm. ACM*, Vol. 28, No. 1, Jan. 1985, p. 8.
7. R.P. Weicker, *Comm. ACM*, Vol. 27, No. 10, Dec. 1984, p. 1013.

Emil W. Brown, a graduate student at Case Western Reserve University in Cleveland, was a staff engineer at Sun Microsystems when this article was written. He developed the microarchitecture of the B5000 integer unit and authored the Sparc simulator. His interests include RISC architecture, system design, and system simulation.

Brown received the MSCS degree from the University of California at Berkeley, and the BEE degree from the Georgia Institute of Technology. He is a member of Tau Beta Pi and the IEEE Computer Society.

Anant Agrawal is Sun's director of hardware. He participated in the design of the B5000 IU microarchitecture and managed the development of the chip set. He currently manages the development of high-end machines. Agrawal received his MSEE from Cornell University and BSEE from M.S. University, Baroda, India.



Trevor Creary, a Sun senior hardware staff engineer, developed the microarchitecture of the B5100 floating-point controller. His interests include system-level CPU design and verification. Creary received the MSCS degree from the Massachusetts Institute of Technology and the BSEE degree from the University of the West Indies. He is a member of the IEEE Computer Society.

Michael F. Klein, a Sun staff engineer, leads the design of an ECL CPU board using the B5000 chip set. His research work at Berkeley concerned the application of artificial-intelligence techniques to computer-aided manufacturing. His interests include computer-aided VLSI and systems design.

Klein received the BS degree in engineering and applied science from the California Institute of Technology in Pasadena and MS and PhD degrees in electrical engineering and computer science from the University of California, Berkeley.

David Murata, also a Sun staff engineer, has worked on the package and circuit design issues of the B5000 IU. His interests include analog and digital bipolar circuit design and simulation. He received a BSEE degree from the University of California at Davis.

Joseph Petolino, a staff hardware engineer at Sun, was the lead designer of the cache memory system for the B5000-based processor described in this article. He designed caches for mainframes while at Amdahl Corp. His professional interests include pipelined processor design and automation of the design and verification process.

Petolino holds an MSEE degree from Stanford University and a BSEE from the Massachusetts Institute of Technology.

Direct any questions concerning this article to Joseph Petolino, Sun Microsystems Inc., 2350 East Bayshore Parkway, Mountain View, CA 94043.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 153 **Medium** 154 **High** 155

Call for Papers

***IEEE Micro* seeks manuscripts for the October 1990 special issue.**

Submit manuscripts to:
Joe Hootman, Editor-in-Chief, EE Dept.,
University of North Dakota, PO Box 7165,
Grand Forks, ND 58202, phone (701) 777-4331.

SOFTWARE AND DIGITAL SIGNAL PROCESSORS

Is it the end of the supercomputer era? Comparison of supercomputer Linpack benchmark price/performance rankings with the latest 32-bit floating-point DSP chips makes serious analysts contemplate the future of the supercomputer industry.

Is DSP software keeping up with the chips? What are you doing to see that it does? *IEEE Micro* would like to feature your DSP system or software efforts. Our readers just can't wait to see what awaits them in the 1990s. Tell us all about it.

Submit manuscripts by March 1, 1990

Hot Chips and Soggy Software

When I sit down at a computer system, I frequently find it difficult to tell what aspects of the system are implemented in hardware, firmware, microcode, and software. I find this particularly true with system performance. So many potential bottlenecks exist—CPU speed, cache size, memory size and latency, bus bandwidth, I/O speed and latency, software algorithms, file system structure, and compiled code efficiency. Unfortunately, performance is a "weakest link" phenomenon; doubling the speed of the CPU won't help if memory or I/O speed already bottlenecks a system.

In the past few years, we've improved the price/performance for CPU cycles at an annual rate approaching 2 to 1. At the same time, we've improved the price/performance for most other system components (displays, I/O speed, networking, software production) much more modestly. In the early 1990s, all system components will be more powerful, but the CPU will be even an order-of-magnitude more powerful.

For decades, CPU performance was a common bottleneck, and the end user immediately noticed improved CPU performance. This extra speed was soon consumed by window systems, relational databases, spreadsheets, Ada, and other software with voracious appetites for CPU cycles. In today's systems, CPU speed no longer automatically forms the major performance bottleneck; the new generation of "hot chips" will provide an abundance of cheaper cycles.

System designs that use cheap CPU cycles to replace slower, more complicated components will have an increasing advantage in the 1990s. Conversely, system structures that were optimal during the 1970s will become increasingly inefficient in the next decade.

What's worked well

By almost any standard, the RISC (reduced instruction-set computer) revolution has had a substantial technical impact on the industry. We've seen the price per instruction per second, which dropped at a rate of roughly 15 percent a year for the previous two decades, drop by more than 50 percent a year over the last several years. The price per floating-point operation shows an even more rapid decline. What did we do right in the RISC revolution?

RISC success springs partially from good system design. Take note and eliminate the software bottleneck from your new design.

Stephen C. Johnson

Stardent Computer Corp.

0272/1732/90/0200-0023$1.00 © 1990 IEEE

Soggy software

One characteristic of all the seminal work on RISC chips is that it was carried out in close conjunction with a strong compiler team. Designers compiled significant programs for "paper" chips early in the design phase. Measurements and simulations identified bottlenecks, and the hardware and software designers together decided how best to eliminate them. This situation existed at several sites. IBM Yorktown Heights used this approach with the 801 machine and the PL.8 compiler, as did the University of California at Berkeley with the RISC chip and an intensive compiler effort using the Portable C compiler. Stanford University also used it with the MIPS chip and an optimizing Pascal compiler. Similar efforts took place at Hewlett-Packard, AT&T Bell Labs, and other universities. These teams also kept up contacts with operating-systems designers and other software developers.

Close design relationships allowed trade-offs to be made that moved problems across the hardware-software boundary. For example, suppose we take the simple matter of how many registers a machine should have. On the purely hardware side, some trade-offs affect this decision—having fewer registers means more space for other components (larger caches) or perhaps a smaller, cheaper design. In addition, designers must consider:

• *Electrical issues.* Fewer registers may mean lower capacitance and higher clock frequencies (if the register bus is a critical path).

Homilies

The following rules of thumb may contribute to better system design:

• **Visualize the whole system.** Know who the end users are. Collect statistics on the important characteristics of the system (interrupts, function calls, memory sizes). Understand the cost constraints of your targeted users and make sure your solutions are consistent with these costs.

• **Write down your assumptions.** These should include your engineering assumptions about hardware and software structure and performance, the model of the end user, and your assumptions about cost. You are unlikely to get these right at first and having them written down may help you make vital midcourse corrections.

• **Collect as many numbers as you can about the behavior of the current design.**

• **Whenever possible, argue about these numbers, not about philosophies.** "It is better to curse the candle than to curse the darkness."

• **Put in the up-front time to learn to talk to people from different disciplines.** Interaction is especially important between hardware and software people, who historically have had profound differences. Today, we see these disciplines blending somewhat. Hardware designers program PALs and use automatic synthesis techniques for gate arrays, while software designers write distributed systems and code to personalize I/O interfaces. We hear hardware and software people talking about the same thing with such different vocabularies that they can't understand each other. It is a poignant mixture of high comedy and low tragedy.

• **Resist the pressure of product cycles.** Many systems have a hardware-debugging stage, which is followed by a software-debugging stage. When the software designers occupy the hot seat, the hardware designers likely have time on their hands. What better time to design the next system! However, the software designers, being pretty busy, can't contribute much. When their feedback on problems comes late in the design cycle, we end up with a perpetuation of previous designs. A little up-front software expertise can save a lot of headaches later.

• **Keep it simple!** Today's hardware technology changes so fast that the product is likely to need a lot of tuning over its lifetime to remain competitive and still take advantage of newer, cheaper systems. Complexity is rarely justified when you look at the total product cycle and really make a sober analysis of what it costs. Getting a product out six months earlier is likely to translate into a 20- to 30-percent performance advantage just by being able to use newer chips sooner.

• **Just say no to options.** A quality product has to be tested. Simple on/off options may double the amount of testing that should be done—everything must be tested with the option on and again with the option off. If you offer an option, be very careful of its span. Try to limit its effect as much as possible, make certain that the interface between the part affected by the option and the rest of the program is as clean as possible, and do a lot of testing at the interface.

• **Validate your assumptions (the ones you wrote down when you visualized your model).** Are the real users behaving like the model suggested? If not, you can get a good hint about where you may have future problems. This hint constitutes a big investment in the future; next time, you'll do it better.

• *Architectural issues.* The bits saved in the register field of the instruction may be used to add more opcodes or address modes, or to decrease the code size of an application.

• *Software issues.* As registers become a scarcer resource, we become more constrained as to their uses.

• *Performance issues.* Fewer registers lead to slower code within a function, with intermediate results "spilled" to memory more often. But fewer registers can also mean lower overhead for function calls and interrupts.

To evaluate proposed changes in an architecture from a systems perspective is difficult. The effect of the change must be reflected in the chip, board, hardware system, compiler, and software system models to understand the impact on the end user. Because designers find this perspective so hard to achieve, we don't see traditional systems that are very well tuned. Even systems that are tuned at one moment in time quickly become suboptimal when we add new chips, memory, and software.

As one simple example, think of the many recent attempts to add X Windows and/or Unix to hardware systems that were not designed for them. In most cases, these attempts went through a period of performing very badly—in some cases, three to 10 times worse than comparably priced systems running the same application. Often, designers needed significant hardware or software tuning to raise performance to competitive levels.

Trends

In addition to the already discussed decreasing cost of cycles, a couple of other trends will affect the optimum design points for future systems.

As chips become faster and more powerful, data movement increases in importance as a critical system bottleneck. Older systems' performance was sensitive to CPU parameters such as the time needed to process an add or multiply operation. For newer systems, performance may be more sensitive to such parameters as cache size and structure, memory and bus speeds, and the quality of register allocation done by the compiler. Data movement instructions often make up nearly 50 percent of all the instructions that a compiler emits.

Traditional compilers pretty much ignore the data movement issue. The programming language conventions establish data layout, and the compiler doesn't mess with it. The studies that accompanied the early RISC machines made it clear that intelligent compiler control of data layout and register allocation could lead to significantly faster programs. Compilers in the 1980s have increasingly produced optimizations that bring data into registers, use it, and put it back only when other parts of the computation need to find it. The 1990s will accentuate this trend. Sophisticated new tools such as dependency analysis allow data references within loops to be analyzed, leading to much better register allocation, cache utilization, and use of special-purpose instructions such as vector operations.

Another trend that is developing rapidly is parallelism. We see not only a rapid growth in multiprocessor systems but also a lot more use of independent coprocessors for floating-point operations and I/O. The Intel i860, for example, contains vectorlike instructions that let it execute integer and floating-point instructions at the same time and process three bus transactions in parallel. Very long instruction word (VLIW) machines gain their power by controlling several independent execution units simultaneously.

Once again, traditional compilers do not address parallelism. To schedule several operations concurrently, compilers must ensure that the data needed by each operation not be changed by the concurrent operations. Dependency analysis develops this information, and a new generation of compilers uses it to produce impressive parallel performance on traditional applications.

These trends relate to each other, not only by impacting each other but also by affecting, and being affected by, the decreasing cost of CPU cycles. At certain price points and for certain applications, the optimal system may have a great deal of parallelism. The data movement between independent elements, and the resulting need for synchronization, may drive a lot of the hardware cost, and may prove to be a major bottleneck unless carefully designed. Unfortunately, even many of the newer chips do not provide multiprocessor support, or their support services only a very narrow range of potential applications. In some cases, designers attempt to retrofit multiprocessor support into existing systems, with indifferent results. Some real opportunities exist for new designs that incorporate system-level multiprocessing support into chips from the beginning stages of design, including not just cache coherency and synchronization but also operating system, interrupt, and context-switching support.

Drying out soggy software

Software productivity has been increasing very slowly over the last decade. In fact, software development techniques could hardly keep up with the historic 15-percent improvement in price/performance per year. Also, software has been left in the dust by the 50-percent-per-year improvements that have characterized hardware in the last few years.

System developers have watched software become a larger and larger part of the development budget, the development time line, and the maintenance work load. The situation reached crisis proportions in 1989, with virtually every major software vendor announcing

significant slips in product delivery. The software industry was frozen by standards fights resembling medieval discussions about angels and pinheads. Investment in a significant new software effort has become so expensive that companies hold back. It is economically disastrous to back the wrong standards horse.

Luckily, the very hardware trends that have driven the problem to a crisis also contain the seeds of the solution. Compared to hardware, software is unstandardized to a degree that horrifies the typical electrical engineer. It seems as if every home appliance were optimized to run only at the precise voltage and frequency that gave optimum efficiency. While we may get the most efficient toaster possible this way, the system-level solution remains unacceptable. Hardware designers, and other engineers, for years have accepted certain standard designs (such as voltage and frequency standards) in the interests of system-level efficiency. Even though a particular appliance may be a few percent less efficient, the cost of electricity can be driven quite low by having only a single kind of power plant, so the total social cost is minimized.

Standardization lets us reuse common components. The toaster and the coffee maker can share a power plant, as well as more mundane items such as wall plugs and wires. We've talked up software reuse for several decades but still honor it more in the breach than in the observance. We write most code today in computer languages that are 15 or more years old and designed for systems with memory and cycles that were 10 to 50 times more expensive relative to other system components than the systems of the 1990s. The low-level reuse mechanisms we do have are terribly constrained by a need to save those precious cycles.

The shape of the future is clear—we can't afford to build software by throwing waves of expensive programmers into the breach armed with methods designed to save cheap CPU cycles. Instead, we must construct successful developments from programs that will appear horribly inefficient to those who cling to the worldview of the past, but will actually be far more efficient in a systems context. They will be cheaper to write, change, and maintain, will be easier to reuse, and will appear in a much timelier fashion than those produced with older methods. Moreover, their very malleability will make it easier to adapt these programs to changes that arise from new hardware, software, or user requirements.

My crystal ball suggests that new methods of building programs (I hesitate to call them languages) will evolve rapidly in the next few years. Rather than focusing on producing discrete mathematical functions of discrete inputs, like most current languages, they will be oriented toward producing and controlling changes in a set of persistent data stored in the computer memory, file system, and network. The tools themselves will be more oriented to making small changes in large programs than to sitting down and writing things from scratch. Reuse will be automatic and invisible, since a large and growing body of program structures will be instantly available. Finally, we'll probably represent these programs two dimensionally and graphically; we may not see any text representation at all.

Electrical engineers will recognize this description; it looks a lot like a circuit! Current CAD tools take a large database of devices and components and allow designers to connect these components, validate them, simulate the result, and finally link the design to manufacture. Like future programs, these circuits are highly parallel; moreover, the system handles most of the low-level details (pin numbers).

New program construction systems will probably have similar features but will also allow building blocks to be adapted from older languages and reused freely. Simulation of components and subsystems will be supported. The compilation process will look more like board or chip layout than like current compilation techniques.

The amount of chaos caused by a new computer language is considerable; the successful ones will have strong ties to the past and preserve the value of much of the current software investment. However, the problem is serious and the philosophical break must be sharp—taking baby steps will not solve today's serious problems.

Stephen C. Johnson is vice president, programming technology at the newly merged Stardent Computer, where he has had a variety of titles including vice president of software. Previously, he worked as head of the Language Development Department at AT&T Bell Laboratories. A researcher there, he wrote the Unix commands, *yacc*, *lint*, and the Portable C Compiler, and cooperated with Dennis Ritchie on the first Unix port.

Johnson graduated from Haverford College and received a PhD from Columbia University in mathematics. He has been on the Usenix Association Board of Directors for six years, the last four as treasurer, and is a member of ACM and IEEE.

Address questions concerning this article to the author at Stardent Computer Corp., 880 West Maude, Sunnyvale, CA 94086.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156 **Medium** 157 **High** 158

The i486 CPU: Executing Instructions in One Clock Cycle

A cache integrated into the instruction pipeline lets this 386-compatible processor achieve minicomputer performance levels.

John H. Crawford

Intel Corp.

Intel began the i486 processor development program shortly after it introduced the 386 processor in 1985. From initial concept, the chip design team worked with the following CPU goals:

1) ensure binary compatibility with the 386 microprocessor and the 387 math coprocessor,
2) increase performance by two to three times over a 386/387 processor system at the same clock rate, and
3) extend the IBM PC standard architecture of the 386 CPU with features suitable for minicomputers.

Compatibility. We recognized the need to be compatible with the 386 Family Architecture.[1-3] As Table 1 shows, a large base of installed software exists to run on a processor that can execute this instruction set. This need

Table 1.
The 386 Family Architecture software base.

	Operation system	
DOS	Unix/386	OS/2
10,000+ applications	3,000 applications	Multitasking
150+ 32-bit applications	Full 32-bit operating system	Graphical interface
Windows graphical interface	System V, Release 4.0	Multithreaded applications
Windows/386 multitasking	Multiprocessor Unix	LAN+ database

0272-1732/90/0200-0027 $01.00 © 1990 IEEE

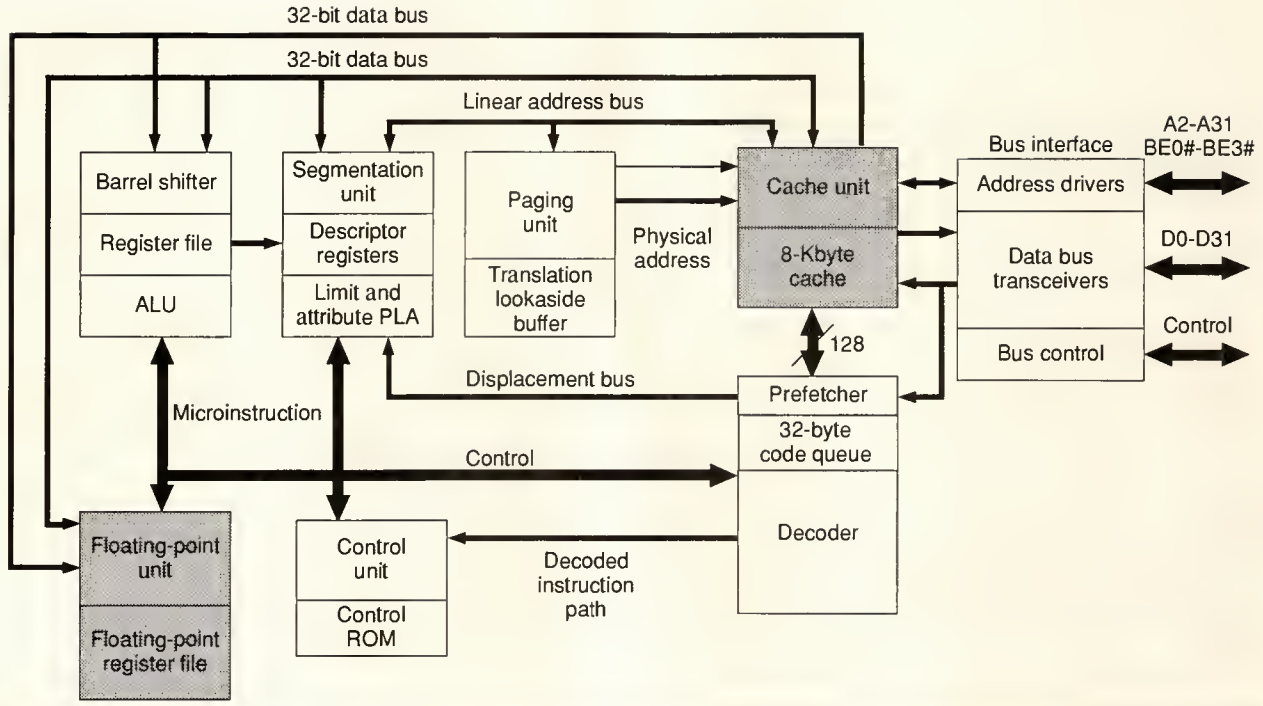


Figure 1. Block diagram of the i486 processor.

to be compatible with an existing software base influenced many of the design decisions during the development of the i486 processor. Where appropriate, I note these influences throughout this article.

In contrast to the clear choice for software compatibility, we saw no strong need for hardware compatibility, that is, to provide the same hardware interface (bus) used in the 386 CPU. Indeed, to achieve higher performance at the same clock rate, and to provide higher clock rate selections over time, the i486 CPU bus had to differ significantly from the 386 processor bus. Though I don't discuss the bus here, Intel's data book describes it in detail,[4] and Grochowski and Shoemaker's article[5] discusses our bus design decisions.

Performance. We used a 1.5-micrometer process to fabricate the 386 CPU, which contains approximately 275,000 transistors. However, we developed the i486 CPU for a 1-μm CMOS process that provided more than two times the number of transistors, or 1.2 million. The process also provided a dense static RAM cell suitable for an on-chip memory cache. The significant increase in the transistor budget provided the raw material for performance improvement.

Figure 1 shows the block diagram of the i486 processor. The shaded blocks indicate functions that are partitioned onto separate chips in a system based on the 386 CPU but which appear on chip in the i486 CPU. These functions are

- the FPU (floating-point unit), which boosts performance of floating-point operations; and
- the 8-Kbyte, unified cache memory.

Although the remaining blocks look similar to the corresponding blocks in the 386 CPU, note that the underlying logic is quite different. We revamped the prefetch, instruction decode, and control logic to optimally integrate the cache into the instruction execution pipeline. This critical improvement allowed the i486 CPU to execute the most frequent, simplest instructions in one clock cycle.

Note also that, like the 386 processor, the i486 CPU includes all the hardware necessary for a complete memory management and protection mechanism. This mechanism provides four protection levels using a combination of segmentation and paging techniques. An on-chip paging cache, or TLB (translation lookaside buffer), of 32 entries supports paging operations.

Let's look at the cache and the integer pipeline. To be brief, I won't discuss the floating-point pipeline here.

On-chip cache

Although the performance goals required, and the transistor budget supported, an 8-Kbyte, on-chip memory cache, we needed to explore a large design space to obtain the best cache design for the i486 processor. The

need to be compatible with the existing software focused the design to a cache that would be invisible to software. In particular, the hardware had to guarantee consistency of the cache with main memory without software intervention.

It may not be obvious that a lowly PC needs any cache consistency support. However, even the original IBM PC (1981) includes a DMA (direct memory access) processor that can generate writes to memory by "stealing" the memory bus from the CPU. Also, we knew the 486 CPU had to execute old versions of operating systems (DOS, OS/2 version1.1, Unix) that did not/could not include any "cache flush" instructions.

We achieved cache-consistent hardware support with a buffered write-through consistency protocol combined with "snooping" (or address detection) hardware. We also added a pin to allow external address-decoding hardware to identify noncacheable memory areas. As an option for new versions of operating systems that use paging, we included new attribute bits in the page table to identify noncacheable memory at a page granularity.

Buffered write through. A buffered write-through technique keeps main memory consistent with the contents of the on-chip cache by passing all store operations (memory writes) to the main memory. All memory writes initiated by the CPU move directly to main memory, a flow that is smoothed by four buffers. This process ensures that the cache and memory remain consistent despite writes to memory by the processor.

While writing to main memory typically takes several clock cycles, the CPU executes a store instruction in one cycle. Without buffers, a store following another store would have to wait for the write operation to complete to memory before it could execute. In a system with a well-tuned memory system that completes a write cycle in two or three clock cycles, the four write buffers permit up to six consecutive writes to occur before introducing a stall. The first write will complete before the write buffers fill up, permitting the second write to be pulled from the write buffers before the sixth consecutive write occurs. This feature is particularly important for procedure calls that typically contain a sequence of store operations to push parameters to the stack, push a return address for the call, and save working registers by pushing them onto the stack.

Consistency in cache and main memory also requires correct handling of writes to memory that are initiated by bus masters other than the processor (DMA or a second processor). The snooping hardware on the CPU handles this aspect of consistency. When a bus master other than the i486 processor performs a memory write, the system hardware provides the address at the pins of the processor. It also activates a control pin to indicate that the address at the pins should be invalidated if present in the on-chip cache. Upon activation of this control pin, the CPU will sample its address pins and invalidate the line mapping that address, if present in the cache. If the address is not found in the cache, the state of the cache does not change. In this fashion, the system hardware can force the processor to remove from the cache any areas of memory that are written by other bus masters.

Cache parameters. We fixed the cache size at 8 Kbytes based on the limits of the 1-μm process used for the i486 CPU. We selected the other cache parameters to maximize the effectiveness of the cache, given this fixed size.

We implemented the cache using a four-way, set-associative organization[6] with LRU, or least recently used, replacement technique. To interleave cache and TLB access, we needed at least a two-way, set-associative cache to achieve an 8K size, since we had a 4-Kbyte page size. A four-way, associative design would provide a 10-percent lower miss rate[7] than a two-way cache yet require minimal additional die area. An eight-way design, however, would provide little improvement.

Based on performance simulations of the cache with address traces captured from a 386 CPU,[5] we used a 16-byte line size. We felt it necessary to perform simulations specific to the i486 CPU since we had found little published information on the overall performance of a cache.[7,8] In addition, we realized the proper optimization is very dependent on specific implementation details. These simulations indicated that a larger line size (32 bytes) would actually slightly decrease the performance of the CPU, even though it would provide a higher hit rate in the cache, a phenomenon also reported in the last two references.[7,8]

The cache, bus, and CPU core minimize the impact of a cache miss on performance. Thus, we achieve the maximum performance possible given a fixed-size cache, and the design supports the use of a second-level cache external to the CPU for maximum performance configurations. Our main considerations were to

1) *Use a burst cycle for cache misses to retrieve a cache line from memory with the minimum bus occupancy.* The burst cycle requires the normal latency to access the first 32-bit piece of a line, but the remaining three pieces can be retrieved at the rate of one per clock cycle. Once the first piece is addressed, the line can be read in with a "burst" of data cycles.

2) *Structure the bus and cache so that burst cycles first access the data causing the miss.* This feature requires that a burst cycle be capable of starting on any one of the four 32-bit pieces of our 16-byte cache line.

3) *Structure the CPU and cache so that the CPU can proceed with the next instruction(s).* This feature requires the CPU to proceed once the data that caused the miss has been retrieved, while the cache and bus complete the line fill in the background.

4) *Use a line buffer for cache fills.* Here, we wanted to take only one cache cycle for cache fills, rather than

Table 2.
Miss rates in the i486 CPU memory cache.*

Program	Miss rates			
	4-byte prefetches	16-byte prefetches	Read	Total
1-2-3	—	0.00	0.02	0.01
Auto CAD	—	0.02	0.01	0.02
Windows 386	—	0.05	0.11	0.08
Frame	0.05	0.14	0.06	0.10
Sunview	0.04	0.10	0.08	0.09
Falcgph	0.02	0.06	0.09	0.07
Falcsnd	0.04	0.11	0.07	0.09
Invframe	0.07	0.20	0.08	0.14
Tpascal	0.04	0.11	0.05	0.08
Troff	0.02	0.06	0.02	0.04
Geometric mean	0.04	0.09	0.07	0.08

*1-2-3 and AutoCAD not included in mean.

four. By doing so, we free up cache cycles for instructions following a miss.

Cache performance. Trace-driven simulations provided a confirmation of our choice of cache parameters. Table 2 lists the miss rates for traces from several single-user multitasking environments, resulting in an overall 8-percent miss rate. This table includes data for 4-byte prefetches to correlate with other published data, as well as for 16-byte prefetches that are more useful for estimating the performance of the cache integrated into the CPU. The 8-percent overall miss rate correlates closely to the 8.4-percent "design target miss rate" cited by Hill and Smith.[7,10]

Cache bandwidth. Because the instruction pipeline can execute most instructions, including those that reference memory, in one clock cycle, the cache on the i486 processor must support a peak rate of two memory references per clock cycle. The references use 1 to 8 bytes to load/store a data operand for the current instruction, and 1 to 15 bytes to fetch an instruction further ahead in the instruction stream.

Two approaches provide a good solution to the need to perform two memory accesses in one cycle. One approach splits the cache into an instruction cache and a data cache, and provides each cache with a private bus to connect to the processor core. The other approach uses one cache to store both instructions and data. It also uses an instruction prefetch buffer with a wide bus to the cache to minimize the contention at the cache between instruction and data accesses.

Many microprocessor implementations use the split-cache approach.[11-14] This approach is particularly attractive for implementations that have the caches external to the CPU, since pin limitations dictate that narrow buses connect to the cache. It is particularly effective when processor pins can be shared by time-multiplexing the signals for the instruction and data buses.[14] In most RISC processors, the cache ports can be 32 bits wide, since all instructions and the majority of data references (except for programs making heavy use of double-precision floating-point data) are 32 bits wide. This fixed partitioning guarantees that the peak rate of one instruction fetch and one memory access can occur in one clock cycle with no contention.

The unified-cache approach using a wide bus in conjunction with prefetch buffers provides an alternative that is particularly attractive for the i486 CPU. The 386 Family Architecture requires a prefetch buffer to support fast alignment and decoding of variable-length instructions. It is easy to support a wide path from the on-chip cache into this prefetch buffer with on-chip wiring. In fact, the chip plan places the prefetch buffers close to the cache array to minimize the wiring required. This plan provides a very high bandwidth path for fetching instructions, thus minimizing the contention of instruction and operand access to the unified cache. A 16-byte bus gave us a bandwidth of 400 Mbytes/s (at 25 MHz) to fetch instructions. Since the i486 CPU requires approximately 50 Mbytes/s of bandwidth on the average, we minimize the contention between instruction and operand accesses to the cache.

The simplicity and performance of the unified-cache approach also influenced our decision. A unified cache simplified the design in several ways. For example, we had to design only one cache! Also, the prefetch buffers needed for this scheme were required anyway to support high-speed decoding of the 386 Family Architecture instructions. Third, no special logic is required to handle self-modifying code.

The unified cache improves performance in ways that compensate for the small amount of contention that remains between instruction and operand access to the cache. 1) The cache accommodates an 8-byte operand access in one clock cycle. This feature improves the performance of instructions referencing double-precision floating-point data and instructions that load segment descriptors. Both require an 8-byte operand access. 2) The hit rate of the unified cache is higher than separate caches of the same total size.[7,9,10]

The improved hit rate results from the lack of fixed partitioning between instructions and data. Consequently, the partitioning can adjust itself based on the instantaneous locality of code versus operand references. The charts in Figure 2, which use a "stacked-line" graph to show the contents of the cache over time, illustrate this phenomenon. In these charts, the number

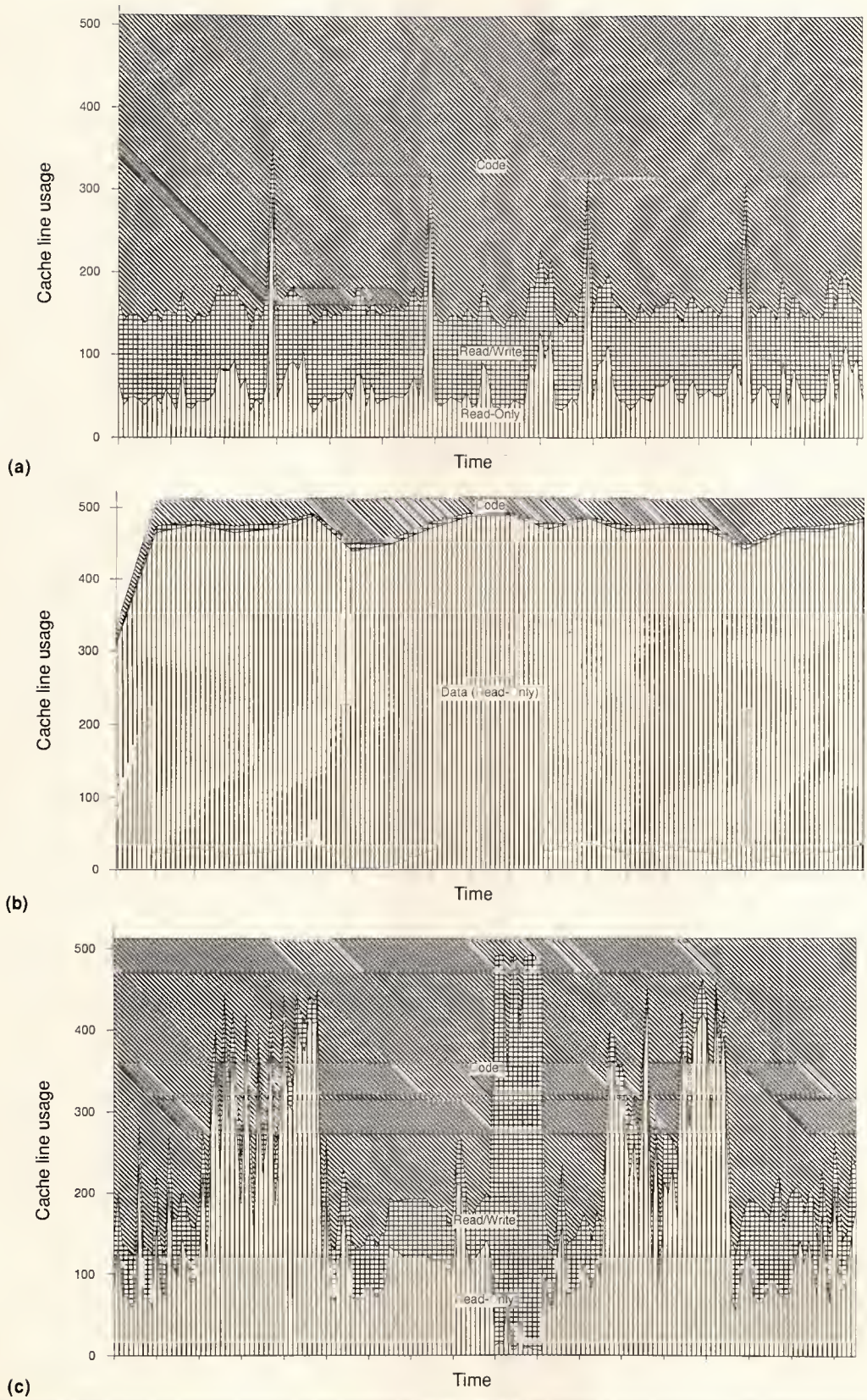


Figure 2. Cache partitioning vs. time: more code than data (a), more data than code (b), and high variability (c).

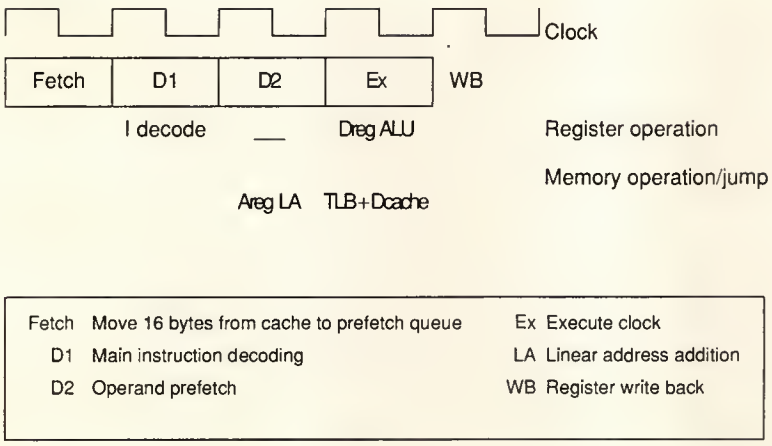


Figure 3. The 486 CPU pipeline.

of lines (out of a total of 512) containing read-only data appear on the bottom, the number of lines containing read/write data in the middle, and the number of lines containing instructions on the top. A cache simulator processing several large address traces taken from a variety of applications produced the charts. Each graph represents 16 million memory references, or about 1 second of activity on a 25-MHz i486 CPU.

Figure 2a shows an application that devotes about 25 percent of the cache to data lines, and 75 percent to instructions. Apparently this application has less locality in instruction accesses than data accesses. In contrast, Figure 2b shows an application that devotes almost all of the cache to data and only a few lines to instructions. Apparently this application is in a tight code loop. Figure 2c shows an application that makes dramatically different demands on the cache during the 1-second sampling window. These examples visually confirm the "self-adjusting" phenomenon that permits a unified cache to exhibit a higher hit rate than split caches of the same total size.

Instruction pipeline

We designed the instruction pipeline to execute instructions at a sustained rate of one per clock cycle. A key to achieving this rate was integrating the cache into the pipeline.

Another key to achieving this rate was pipelining the instruction decoder into two stages to provide a sustained throughput of one instruction per cycle. An overall design concern centered on the fact that existing programs had to perform well. That is, we could not rely on a compiler to cover any inefficiencies in the pipeline.

Pipeline overview. Figure 3 illustrates the five stages in the execution pipeline. In this figure, the horizontal axis represents time.

The Fetch stage extracts the instruction from the cache. D1 represents the main instruction decoding stage, while in the D2 stage secondary instruction decoding and memory address computation take place. In the Ex stage the instruction begins to execute, and in the WB stage results are written to the register file.

Fetch stage. Since the CPU fetches an entire cache line from the cache, most instructions don't require this stage. On the average, the CPU fetches about five instructions during each 16-byte cache access. We include this stage as part of the "average" instruction handling because it is always required at the target of a branch, and of course an instruction must be fetched before it can be decoded or executed.

D1 stage. This first instruction decoding stage processes up to 3 instruction bytes. It also decodes the actions to occur during the D2 stage for computing memory addresses. The D1 stage determines the length of the instruction and directs the instruction aligner/prefetch queue to step to the next instruction, possibly stepping over data to be used in the D2 stage.

Decoding of instruction prefixes proceeds 1 byte at a time, with each prefix occupying a single D1 stage in addition to the main D1 stage for decoding the opcode. Instructions having a 2-byte opcode, those containing hex 0F in the first byte, require two clock cycles in the D1 stage (as if 0F were a prefix byte). All other instructions use just one D1 cycle. About 6 percent of the instructions in a "typical" DOS program require the extra D1 cycle, while the average Unix program proportion is less than 3 percent.

D2 stage. This stage completes the decoding of instructions and also computes memory addresses. It decodes a 1- to 4-byte memory displacement field, or a 1- to 4-byte immediate constant per clock cycle. If both a memory displacement and an immediate constant are present in the instruction, decoding takes two D2 cycles.

The D2 stage also computes memory addresses in parallel with decoding displacements or immediate constants, as directed by the D1 stage. It will decode a displacement (if present) and add it with a base register (if present) in the first D2 cycle, and a scaled index register (if present) in the second cycle. If no index register exists, address computation completes in one D2 stage. Only about 5 percent of the instructions require two D2 stages.

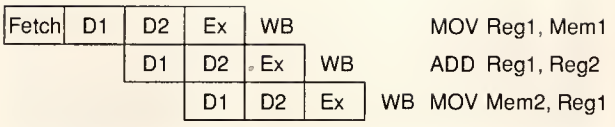


Figure 4. No data load delay in the pipeline.

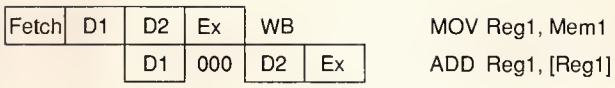


Figure 5. A pointer load delay.

Ex stage. A microprogram in the microcode ROM controls the actions of the main execution stage of the i486 CPU; hardwired logic controls all other stages.

Instructions that reference memory (including jump instructions) access the cache in this stage. Instructions that load data from memory use one Ex clock cycle, upon a cache hit. Cache lookup proceeds in parallel with the TLB lookup phase.

Instructions that perform arithmetic in the ALU using values from on-chip registers or immediate constants enter the Ex stage to read data from registers, compute the necessary function, and latch the results.

WB stage. This stage updates the register file either with 1) data read from the cache or main memory or 2) a result from the ALU output latch. We included the necessary bypass paths to avoid stalling the pipeline should the WB stage of one instruction write to a register that is used in the Ex stage of the next instruction. These stages occur at the same time for the two successive instructions.

No data load delay. Most RISC machines pipeline memory access to an off-chip cache. As a result, the data becomes available from a load instruction too late to be used by the next instruction, even if a cache hit occurs. Instead, the data generally becomes available after a delay of one[12-14] or two[11] clock cycles. This phenomenon, known as a load delay, means that for maximum performance these processors require the compiler to reorder instructions so that unrelated instructions can be inserted between a memory load and an instruction that uses the new data.

We had to take care to avoid a load delay in the i486 CPU, since most of the software to run on the processor exists as "shrink-wrapped" disks developed long before the CPU was available. So, we could not rely on a compiler to make up for any load delays. Instead, we wanted to ensure that data loaded by one instruction could be used in the next instruction. The CPU pipeline supports this requirement, as illustrated in the three-instruction sequence shown in Figure 4.

The first instruction in the figure (the MOV Reg1,Mem1 load instruction) moves data from the memory location given by Mem1 into register Reg1. The second instruction (ADD Reg1,Reg2) adds the contents of register Reg2 to this new value in register Reg1 and leaves the result in Reg1.

The load instruction accesses the cache in its Ex clock cycle. With a cache hit, the data becomes available at the end of the Ex cycle. The CPU writes the data into the register during the WB cycle, which occurs at the same time as the Ex cycle of the add instruction needing this new data. Because the CPU contains the necessary bypass paths, it can write this data to the register file and forward it to the ALU at the same time, avoiding a load delay.

The third (MOV Mem2,Reg1) instruction also has a potential load delay. The value computed in the previous instruction (and stored into Reg1) must be available in the Ex stage of this store instruction so that it can be written into the cache. Again, the data moves along the necessary bypass path from the WB stage of the previous instruction.

The on-chip cache and computation of memory addresses in the D2 pipeline stage eliminated the load delay. Having the cache on chip allows a full cache operation to complete in one clock cycle while minimizing the latency that must be covered by pipelining. By arranging the pipeline so that this one cycle of cache lookup occurs in the Ex stage of the pipeline, we eliminated the load delay. To arrange the pipeline this way required that memory addresses be computed before the Ex stage; the CPU computes them in the D2 stage. As this D2 stage was needed anyway for pipelining the instruction decoding, we eliminated the load delay without adding extra stages to the pipeline. Processors with off-chip caches require at least 1.5 cycles for a cache access and typically don't require our extra decoding stage. As a result, they require a load delay of one or two cycles.

Pointer load delay. Although the CPU does not have a load delay for data operations, it does require a load delay for values used to compute memory addresses. That is, if a value is loaded from memory into a register, and that register is then used as a base register in the next instruction, the CPU will stall for one cycle. The delay occurs because of a pipeline collision between the WB stage of the load and the D2 stage of the instruction using the base register. These stages collide because the CPU pipeline computes memory addresses during the D2 stage, requiring that the base register be accessed one clock cycle earlier than if it were used as data, as in the previous example.

Figure 5 illustrates the pointer load delay with a two-instruction sequence that loads a value from memory into a register. It then uses the register as the base

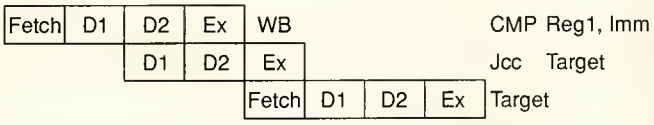


Figure 6. Jump instruction timing.

Table 3.
Clock cycle counts for basic instructions.

Instruction type	Counts 386	486	Sparc	88100
Load	4	1	2	1-3
Store	2	1	3	1
ALU	2	1	1	1
Jump taken/	9	3	1	2
not taken	3	1	2	1
Call	9	3	3	1

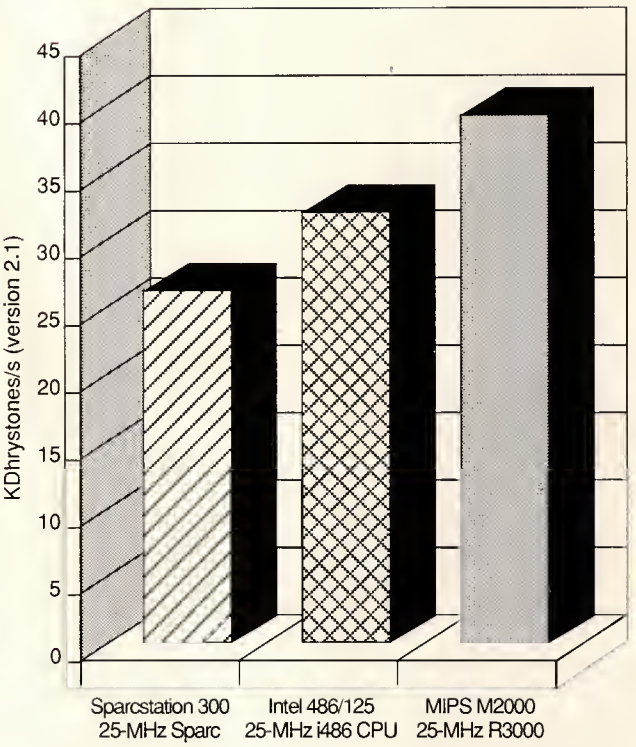


Figure 7. Selected processor performance results from Dhrystone version 2.1 tests.

register in the next instruction, which also is a load instruction. In this case, the CPU accesses the cache in the Ex stage of the first instruction and stores the value retrieved (assuming a hit) in the register during the WB stage. However, the next instruction needs this register in its D2 stage, requiring the CPU to stall for one cycle. When the D2 stage lines up with the WB stage of the previous instruction, the WB-to-D2 bypass path can be used to provide the data for the memory address computation.

Jump instruction timing. Figure 6 illustrates the timing of a jump instruction, assuming that the jump is taken—that is, assuming that the CPU evaluates the jump condition as true. The first instruction in the sequence is a compare instruction, which sets the condition code to condition the jump. The compare operation occurs in the Ex stage, and the CPU writes it to the condition flags in the WB stage. A bypass path makes the condition flags available in the Ex stage of the next instruction, which is a conditional jump. During the Ex stage of the jump, the CPU evaluates the condition to determine if the jump should be taken.

In parallel, the CPU runs a speculative fetch cycle to the target of the jump during the Ex stage of the jump instruction. If the CPU determines a false jump condition, it discards this prefetch and continues execution with the next sequential instruction (already fetched and decoded). In this case (a false jump condition), the jump executes in just one Ex cycle.

If the CPU evaluates the condition as true, the fetch to the target address refills the pipeline. As shown in Figure 6, this means that the Ex stage of the jump instruction corresponds to the Fetch stage of the target. By lining up the pipeline stages of the two instructions, we see that the target instruction reaches the Ex stage three clock cycles after the jump.

In summary, conditional jumps take one clock cycle for a jump that is not taken, and three cycles for a taken jump. Unconditional jump and call instructions also take three cycles.

Performance summary. Table 3 summarizes the clock cycle counts for the most frequently executed instructions for the 386, i486 CPU, Sun Microsystems Sparc,[12] and Motorola 88100[11] processors. Note that the i486 CPU achieves a reduction in cycle counts of between two and four times over the 386 CPU. The i486 CPU does a little better than the RISC processors for data-manipulation instructions but executes a little slower on branches.

Figure 7 shows the relative performance of these processors on the popular Dhrystone benchmark version 2.1. An Intel System 486/125 (Multibus II, 8-Mbyte memory, no external cache, Unix System V release 3.2) running the benchmark as compiled by the MetaWare C compiler, beta release 2.2c, produced the

i486 CPU numbers. We took the Sun Sparcstation 300 and Mips M2000 numbers from published benchmark reports.[15, 16]

The fastest available (at the time of this writing) computer systems containing the various processors produced the benchmark results. Several vendors, including Intel, have announced future availability of faster versions of all of the processors in this chart. However, we had to restrict the benchmark exercise to complete systems available for running the benchmarks.

Figure 8 shows the relative performance of the i486 CPU to several RISC processors on the C programs in the SPEC benchmark suite.[17] This suite contains 10 programs intended to measure the performance of a workstation running "typical" engineering workstation activities. Four of these programs are written in C and emphasize integer performance. They therefore provide a good measure of the effectiveness of the integer performance techniques discussed in this article. The remaining six programs in the SPEC suite are floating-point-intensive Fortran applications not discussed here.

A Compaq Deskpro 486/25, Model 650 (25-MHz i486 CPU, 128-Kbyte cache, 16-Mbyte RAM) running ISC Unix V release 3.2 and using the MetaWare C compiler release 2.2c produced the numbers for the i486 CPU. (Compaq's SPEC license number is 136.) These results—obtained on preproduction hardware and a beta release of the MetaWare compiler—are preliminary and may not reflect the actual results received when the SPEC benchmark runs on a production-level system. By the time this article appears in print, vendors expect production quantities of the system and compiler to be available.

The numbers for the Sun Sparcstation 330 and Mips M2000 systems appeared in the first issue of the *SPEC Newsletter*.[17] Note that all three of the systems ran at a 25-MHz clock rate.

These benchmark results confirm the performance numbers implied by the instruction clock cycle counts listed in Table 3. They show the performance level of the i486 CPU to be competitive with current RISC-based systems.

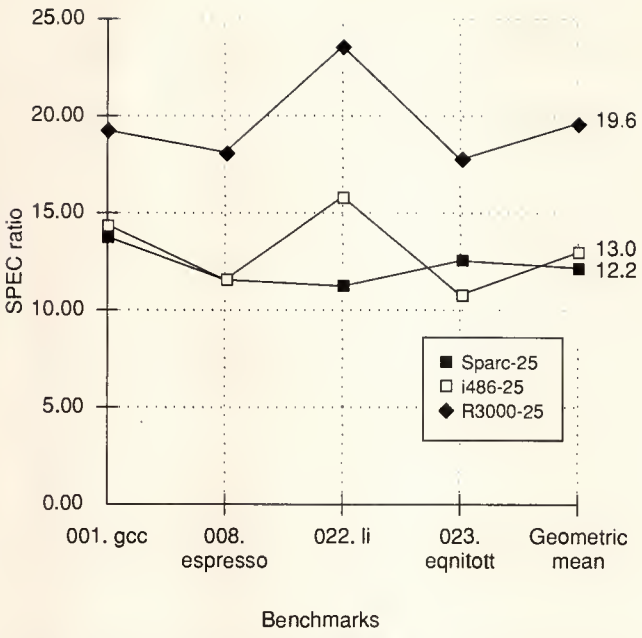


Figure 8. Performance results from C programs in the SPEC benchmark suite.

Although the implementation of the i486 CPU resembles in many ways several popular RISC implementations, certain key decisions strongly affected our final design. Our need to be compatible with the PC standard instruction set and our highly integrated implementation drove these decisions. They are:

1) Hardware had to maintain cache consistency, with no operating system intervention.

2) We chose a unified cache (code and data combined) rather than having separate caches for code and data.

3) Integrating the cache on chip allowed us to avoid a data load delay, an important design factor.

The i486 CPU provides a performance level similar to today's RISC processors yet retains compatibility with a large base of existing software. We accomplished this level of performance along with compatibility by carefully integrating a cache into the instruction pipeline to achieve a sustained execution rate of one instruction per clock cycle.

Acknowledgments

The i486 CPU demanded a very large effort, involving many groups of people throughout Intel Corporation. For the implementation choices just described, I would especially like to recognize Beatrice Fu, Pat Gelsinger, Ed Grochowski, and Ken Shoemaker from the chip design team. I thank Compaq Computer Corp. and MetaWare Inc. for their timely assistance in producing the SPEC benchmark numbers.

References

1. J. Crawford and P. Gelsinger, *Programming the 80386*, Sybex Books, Alameda, Calif., 1987.
2. Intel Corp., *80386 Programmer's Reference Manual*, Santa Clara, Calif., 1986.
3. J. Crawford, "Architecture of the Intel 80386," *Proc. ICCD 86*, Oct. 1986, pp. 155-160.
4. Intel Corporation, *i486 Microprocessor* (data book), 1989.
5. E. Grochowski and K. Shoemaker, "Issues in the Implementation of the 486 Cache and Bus," *Proc. ICCD 89*, IEEE Computer Society, Los Alamitos, Calif., Oct. 1989, pp. 193-198.
6. A.J. Smith, "Cache Memories," *ACM Computing Surveys*, New York, Vol. 14, No. 3, Sept. 1982, pp. 473-530.
7. M. Hill and A.J. Smith, "Evaluating Associativity in CPU Caches," Tech. Report 823, Computer Science Dept., University of Wisconsin, Madison, Feb. 1989.
8. S. Przybylski, M. Horowitz, and J. Hennessy, "Performance Tradeoffs in Cache Design," *Proc. 15th Ann. Int'l Symp. Computer Architecture*, May 1988, pp. 290-298.
9. A.J. Smith, "Line (Block) Size Choice for CPU Cache Memories," *IEEE Trans. Computers*, Vol. C-36, No. 9, Sept. 1987, pp. 1063-1075.
10. A.J. Smith, "Cache Evaluation and the Impact of Workload Choice," *Proc. 12th Ann. Symp. Computer Architecture*, June 1985, pp. 64-73.
11. *MC88100 User's Manual*, Motorola Inc., Phoenix, Ariz., 1988.
12. *MB86900 High Performance 32-Bit RISC Processor Data Sheet*, Fujitsu Microelectronics Inc., Santa Clara, Calif., July 1987.
13. G. Kane, *MIPS R2000 RISC Architecture*, Prentice Hall, Englewood Cliffs, N. J., 1987.
14. *PaceMips R3000 32-Bit, 25MHz RISC CPU with Integrated Memory Management Unit* (data sheet), Performance Semiconductor, Sunnyvale, Calif., 1989.
15. *Performance Brief*, Mips Computer Systems, Inc., June 1989.
16. *DEC Benchmark Report*, Digital Equipment Corp., Maynard, Mass., July 11, 1989.
17. *SPEC Newsletter*, "Benchmark Results," Vol. 1, Issue 1, Waterside Associates, Freemont, Calif., Fall 1989.

John H. Crawford, the chief architect of the 386 and i486 microprocessors, manages the 386 Family Architecture for Intel Corp. in Santa Clara, California. He has been involved with the 386 from its inception in 1982 and previously worked in compiler development. His research interests include high-performance CPU design, multiprocessor systems, computer architecture, microprogram verification, and compiler technology.

Crawford received a BS degree from Brown University and an MS from the University of North Carolina, Chapel Hill, both in computer science. Author of several papers on compiler technology and microprocessor architecture, he also coauthored *Programming the 80386*. He is a member of the IEEE and serves on the Editorial Board of *IEEE Micro*.

Readers may direct questions concerning this article to the author at Intel Corp., 3065 Bowers Avenue, Santa Clara, CA 95051.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 209 **Medium** 210 **High** 211

Compiler Challenges with RISCs

When reduced instruction-set computers were beginning to flower in the early 1980s, one of the common justifications for building RISCs was that their compilers were simpler than those of complex instruction-set computers. Patterson and Sequin argued for the simplified instruction set in an article describing the seminal RISC work at the University of California, Berkeley. This simplification implies that a compiler has to perform only simple case analysis for code generation. They conclude that "simple case analysis implies a simple compiler."[1] An amusing 1984 advertisement for a RISC brags about how "[CISC] parameter passing schemes worthy of a doctoral thesis are replaced by simple [RISC] hardware."[2] Although I am an enthusiastic supporter of RISCs, I hope the reader wonders at the conclusion of this article whether the words *CISC* and *RISC* should be reversed in that sentence.

For simple operations like addition and subtraction, it is true that RISCs generally offer the compiler few alternatives and thus keep the decision-making process simple. But compilers do much more than handle addition and subtraction. Consequently, writing a code generator that takes full advantage of a RISC's potential involves a considerable number of challenges.

The main reason for the challenges is an insatiable thirst for speed. A RISC manufacturer will do almost anything to the hardware for speed, even at the cost of considerable compiler design effort. A CPU with multiple functional units that can operate in parallel usually requires careful scheduling of instructions to keep the units as busy as possible. Poor scheduling results in idle functional units and slower execution. However, a CPU that automatically schedules instructions becomes too complex, slows down overall chip development, and has a number of bugs. It's also probably slower.

RISCs may possess simplified instruction sets, but don't be fooled. Crafting their compilers is not so simple.

Thomas J. Pennello

MetaWare Incorporated

0272-1732/90/0200-0037$01.00 © 1990 IEEE

```
void printf(char * format, ...) {
    va_list ap;
    va_start(ap,format);
    do {
        switch (next-format-indicator) {
            case %s:
                char * sval = va_arg(ap,char *);
                etc.
            case %i:
                int ival = va_arg(ap,int);
                etc.
            case %f:
                double fval = va_arg(ap,double);
                etc.
            }
        etc.
        } (while) (!done);
    }
```

Figure 1. A skeletal printf function illustrating access to variable arguments.

For speed, RISC code should keep variables out of slow memory and place them into fast registers as often as possible. This approach, however, implies a procedure-calling convention that passes arguments in one or more registers. The existence of multiple functional units often implies more than one register file so that speed isn't sacrificed due to contention for a register file. RISCs commonly have one set of registers for floating-point calculations and another for integer calculations. But this feature can mean passing arguments in both register files, with the attendant complexity. Furthermore, the arguments might not all fit in the registers, which requires the compiler to pass arguments in both registers and memory.

Because the branch instruction is anathema to speed, compilers must chose branch-free sequences of code wherever possible, even if longer instruction sequences result.

I am not arguing that RISCs should be made more complicated to ease the compiler writer's job. I am just illustrating the fact that RISC vendors depend heavily upon good compilers to make effective use of the hardware.

This article offers a concrete example of the complexity of writing a compiler for a RISC. I draw examples from personal experience in crafting a code generator for the Intel i860. One of the problems was designing a procedure-calling convention. Because real programs contain a great number of procedure calls, a calling sequence should be fast. Designing a calling sequence exposes many subproblems, such as

- how to pass arguments to a procedure,
- which registers to save across procedure calls,
- how to allocate local stack storage upon procedure entry,
- which register to use as the frame pointer and where it points, and
- how to communicate the return address to the procedure.

Here, I focus solely on the first subproblem—that of passing arguments. A constraint of the solution is that the ubiquitous print function printf (from the C programming language) has to work. I define this problem and show solutions for CISC and RISC architectures.

The Varargs problem

The function printf can take any number and kind of arguments of varying types. Only the first argument—the *format string*—is fixed. Within the format string *format indicators* inform printf of the number and type of the remainder of the arguments. A format indicator starts with a % character. For example, the format string

"Argument number %i is %s"

indicates that the second argument is an integer (%i) and the third (and final) argument is a string (%s). Here the arguments are two different types, but they could also be different sizes (for example, a double-precision floating-point argument is generally bigger than an integer value). The variable arguments (Varargs) problem is how to provide a way by which printf can obtain the successive arguments after the format string.

The American National Standards Institute C Programming Language Draft Proposed Standard[3] prescribes a method for printf to declare its Varargs intent, as well as for picking up the additional arguments, as shown in Figure 1.

This program segment (written in pseudo-C code for brevity) shows the salient characteristics of the method. The ellipsis (...) in the function header indicates that printf can take variable arguments. The construct va_list ap; declares an *argument-list pointer* (ap) that can be thought of as pointing to the next argument. The statement va_start(ap,format); initializes the ap to point to the argument following the format argument. Each call to the macro va_arg needs the ap and the type of argument desired. The macro produces the desired argument and adjusts the ap as a side effect to point to the next argument.

Although C's printf only takes arguments of the types string (char *), integer (int), and floating-point double-precision (double), let's spice up the example a little by adding arguments of the types structure (struct) and floating-point single-precision (float). The function printf isn't the only one granted the Varargs status. Any programmer can write a function and have it take the desired argument types.

The example for the remainder of this discussion is an invocation of a hypothetical Varargs function func. The top section of Figure 2 shows the call to func and

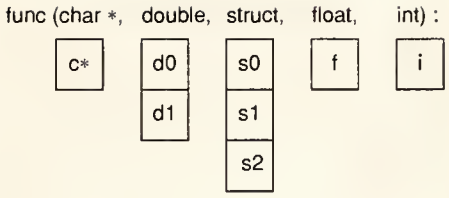


Figure 2. The types and sizes of the five arguments to the call func.

the argument types that are passed. The box sizes relate to the sizes of the arguments. For example, the struct argument occupies 3 words. (Henceforth, a "word" means 32 bits, the size of each register in all examples.) The first char * argument is presumably a format string that indicates the types of the other arguments.

Varargs and CISCs

Most CISCs (80X86s, 680X0s, and VAXs) typically push arguments onto a stack. The va_list type is simply a pointer, and va_start(ap,format); makes the ap point to the double argument after the format string. The va_arg macro advances the pointer the number of bytes used in the representation of the type (with possible padding added to keep arguments word-aligned). Before the pointer advances, the macro obtains the previously pointed-to value as its result.

Programmers have used this simple technique since the invention of the C programming language, even before ANSI began to standardize the way to program Varargs.

Varargs and RISCs

Here I discuss the application of Varargs to two RISCs and evaluate the implications.

The IBM RT PC. The RT's procedure-calling convention specifies that up to four words of arguments are passed (placed) in registers and the remainder are passed (placed onto) to the memory stack, or argument-overflow area. If an argument is too large to fit entirely in the available registers, the portion that does fit occupies these registers. The argument-overflow area accommodates the rest of the argument. See the structure argument in Figure 3, in which register R5 holds the first word of the structure (s0) and the argument-overflow area holds the remainder (s1, s2). Arguments i and f appear entirely in the argument-overflow area. The first two columns of boxes in Figure 3 show the position of the arguments at procedure entry. The third

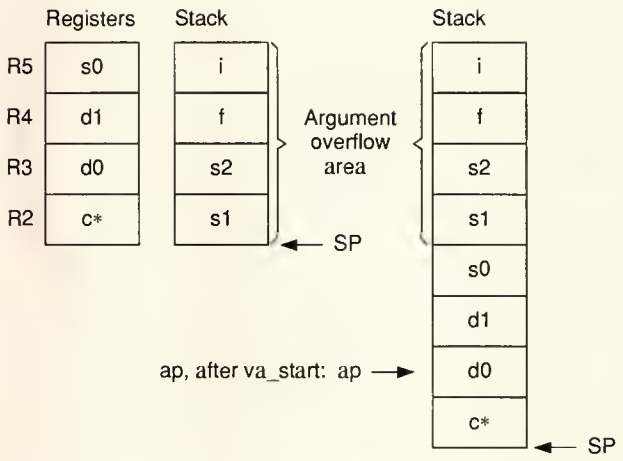


Figure 3. The placement of arguments at the entry to func (two left columns of boxes) and the storage of the register-passed arguments on the stack after the execution of func's prolog code (far right). SP is the stack pointer.

column of boxes depicts the change to the stack that occurs after the procedure prolog and the call to va_start. The ap and va_start of Figure 3 operate in the hypothetical function func in the same way as they do in the skeletal printf of Figure 1. (In all illustrations, the stack "grows down" to decreasing memory addresses, and register numbers at the bottom of a figure are lower than those at the top.)

At procedure entry all register arguments are stored in memory just below the argument-overflow area so the arguments can be addressed. Any arguments that were partially passed in both registers and memory become contiguous, as is the case for the struct argument. This storing is the only added complexity for the RT; the code for va_arg remains the same as in the CISC case.

Intel i860, scenario 1. Here I move on to the i860 and the calling convention published in the programmer's reference manual.[4] The i860 has thirty-two 32-bit integer registers and thirty-two 32-bit floating-point (FP) registers. FP arguments are passed in 12 FP registers; all other arguments are passed in six integer registers. An FP argument consumes two FP registers whether the argument is a float or a double. Each set of registers interferes with the other in that when an integer argument is passed, the two corresponding FP registers are wasted, that is, they cannot contain an argument. Similarly, the use of two FP registers wastes an integer argument register. Wasting registers simplifies the implementation of Varargs. Few procedures have both FP and integer arguments—in fact, few procedures have many arguments. This reasoning justifies the restriction to only six "argument positions."

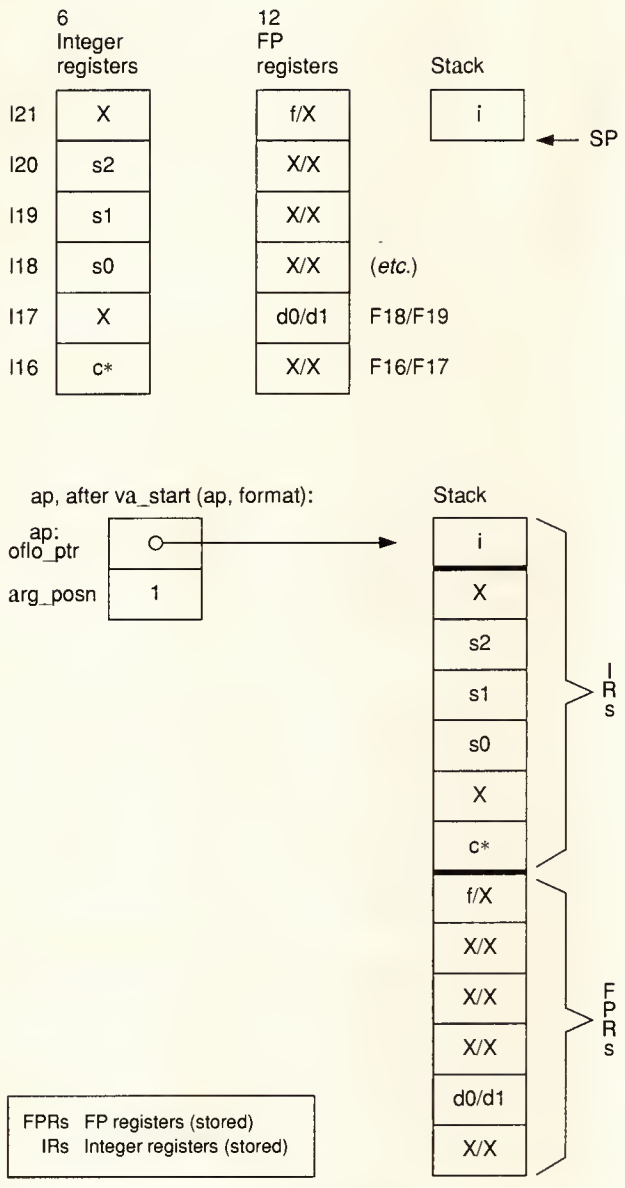


Figure 4. Argument placement and register-argument storage for i860, scenario 1. The "/" mark indicates the depiction of two FP registers in a single box. The "X" indicates a register not used for passing an argument.

Figure 4 demonstrates how the arguments are passed for the function call. The components of ap—arg_posn and oflo_ptr—are necessary for correct operation of the va_arg macro; I explain them in more detail later.

At procedure entry, all register arguments are stored in memory just below the argument-overflow area, as with the RT. The arguments in the integer registers are stored first, then those in the FP registers. The ap can no longer remain a simple pointer because the arguments

```
#define va_arg(ap,type)  _va_arg(&ap, sizeof
  (type),class_of(type))

_va_arg(ap,size,class):
      if (ap->arg_posn >= 6)
         argument is at ap->oflo_ptr[ap->arg_posn-6];
      else if (class == FP)
         argument is at (ap->oflo_ptr-6-12)
           [ap->arg_posn];
      else
         argument is at (ap->oflo_ptr-6) [ap->arg_posn];
      ap->arg_posn +=
       class == FP ? 2 : size/WORDSIZE;
              /* WORDSIZE = 4 (bytes in a word). */)
```

Figure 5. Pseudo-C code illustrating the implementation of macro va_arg for the i860, scenario 1.

are not necessarily contiguous in memory. Based upon its type, a desired argument lies where the integer or FP registers were stored or—as discussed—in the argument-overflow area. The argument pointer has become a structure with two components. The first, oflo_ptr, points to the argument-overflow area. The second, arg_posn, records how many argument positions have been "consumed" up to this time. The value of oflo_ptr doesn't change; only arg_posn varies. (The decision to have oflo_ptr point to the argument-overflow area is somewhat arbitrary. It could point to either the base of the integer or FP storage areas, since both of these are fixed in size.)

For the i860, the macro va_arg requires not only the size but also the class of the argument type (FP or not). Figure 5 illustrates this form of va_arg.

In Figure 5, the −6 (and respectively −6−12) adjusts oflo_ptr to point to the place in which the integer (respectively, FP) registers were stored. The result is treated as an array containing 6 (respectively, 12) words. Because of the simplifying interference convention, one can use just an index, arg_posn, to access the i^{th} argument in either the integer or FP array. Because of the doubling up of FP registers, the i860-required, 8-byte alignment for FP values is automatic. In the case of the specific four arguments to func, the argument type and the amount to increment arg_posn are, in order of argument occurrence:

- double (2),
- struct (3),
- float (2), and
- int (1).

Note that although the C construct sizeof is predefined, the compiler must also supply a helper function (class_of) that takes a type and produces at compilation time a value that indicates FP or not.

The i860, scenario 2. The interference convention and the doubling up of FP registers simplifies Varargs at the cost of wasted registers. However, this waste is unnecessary. Here, I specify a convention that uses 12 integer registers and 12 FP registers for passing arguments. Structures up to 4 words in length are passed in the integer registers. Longer structures are passed in memory—even if integer registers are available—to prevent consuming all integer registers with a single large structure. Further, 1-word float values take up only one FP register and 2-word doubles take two. However, the two must be aligned to an even-odd register pair due to i860 alignment restrictions imposed to obtain speed. The result is more flexibility in passing arguments but more complexity in implementing Varargs. Figure 6 presents the argument-passing picture.

In contrast to the preceding situation, no register wastage occurs and, in fact, seven integer registers and nine FP registers remain free to pass arguments. Again, the integer registers and FP registers are stored in the same way. The ap structure now must be more complex. First, arguments need not necessarily exhaust the register space before they are placed in memory. Structures bigger than 4 words are stored in memory whether or not registers remain for argument passing. Second, integer and FP registers are not coupled and can be consumed independently of one another. This factor mandates the existence of three independent counts (int_cnt, FP_cnt, and mem_cnt) to keep track of the usage of the integer, FP, and argument-overflow areas. The ap structure depicted in Figure 6 indicates that 4 bytes of integer argument area have been used (by the format argument) after the call to va_start.

For va_arg, the location of the argument depends not only on the number of bytes consumed thus far in the appropriate area but also on the alignment. Alignment restrictions may introduce holes in the areas, which can occur when a float argument is immediately followed by a double argument. After obtaining the argument, va_arg increments one of the three counts depending upon the argument's type and considering any alignment holes. In the case of the specific four arguments, the argument type and the ap field incremented are:

- double, FP_cnt (by 8);
- struct, int_cnt (by 12);
- float, FP_cnt (by 4); and
- int, int_cnt (by 4).

Note that two helper functions are required: class_of and align_of. The macro va_arg passes the class and type to the function _va_arg:

```
#define  va_arg(ap,type) \
         _va_arg(&ap,sizeof(type),class_of(type),
           align_of(type))
```

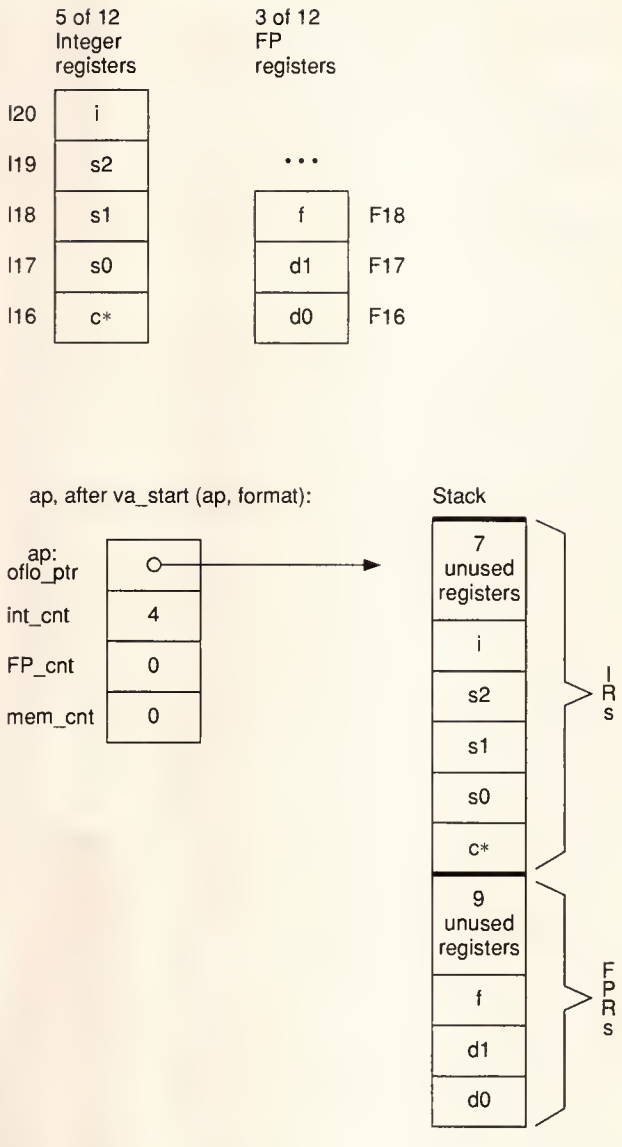


Figure 6. Argument placement and register-argument storage for the i860, scenario 2. No register wastage occurs, and numerous integer and FP registers are available for argument passing.

The actual implementation of _va_arg is a function of about 60 lines of C code, and I will not attempt to reproduce it here. (The address of the ap is taken because _va_arg updates the ap. Were _va_arg a macro, taking the address would be unnecessary.)

In the frequent case of just a few arguments, the cost of a procedure call in scenario 1 or 2 is the same. Scenario 2 just causes the Varargs function to work

harder. But real programs can deviate from this frequent case. (I have seen a time-consuming, military floating-point application that contains a non-Varargs procedure of 22 arguments.) In such cases, the argument-passing convention of scenario 2 is considerably faster. Thus one should always try to obtain the best performance possible. Stated another way, to impose an artificial limit is to ask someone to exceed it.

The last word. The calling convention for the i860 generated a great deal of discussion and concern during its development. The discussion centered around whether an argument-passing scheme as flexible as last described here would really work and how many helper functions it would need. The primary concern was whether compiler vendors could be expected to implement it. MetaWare was the first to implement true Varargs for the i860 and so proved the possibility of doing so. In fact, it implemented both argument-passing schemes. But in recognition of the complexity of the scenario-2 calling convention, the Intel/AT&T-sanctioned application binary interface[5] for the i860 passes no structures in registers (unfortunate for a language such as C++ that lives and breathes structures). The interface makes other simplifying assumptions. It is significant that a somewhat simpler calling convention exists just to make the implementation easier for compiler writers.

As an aside, the reader may wonder why arguments should be passed in registers if all of the examples shown here store all registers in memory and so sacrifice the speed benefits of registers! The answer is that a Varargs function is a member of a rare species, and the overwhelming majority of C functions can take full advantage of arguments passed in registers.

The argument-passing method must always be the same, since the ANSI C Draft Proposed Standard does not require the programmer to distinguish for the compiler's sake which kind of function is being called. Thus it's not possible, for example, to use one method for Varargs and another for non-Varargs functions. This decision was a significant battleground in the C standardization process. Allowing two distinct calling conventions would invalidate the common "hello world" program:

```
main () {
  printf("Hello world.\n");
  }
```

because the program contains no declaration telling the compiler that printf is a Varargs function. (The ANSI C Draft Proposed Standard provides a way to do this. Because invalidating "hello world" is completely unacceptable, the Varargs burden fell solely on the callee side.)

Implementing a calling convention for a RISC is by no means the trivial exercise that it is for most CISCs. But to realize the full potential of a RISC, the complexity of the convention must remain. Consider again the advertisement: "[CISC] parameter passing schemes worthy of a doctoral thesis are replaced by simple [RISC] hardware." Perhaps the phrase simple RISC hardware is accurate, but simple RISC software is not. The code in the MetaWare i860 compiler required for add and subtract operations—where RISCs make life simple and uniform—takes only about 20 lines of C. But the code devoted to stack-frame setup alone is about 400 lines. The numbers are still in the 100s for MetaWare's code generators for other RISCs such as Sun Microsystem's Sparc, IBM's RT PC, and Advanced Micro Devices' Am29000.

In the introduction, I mentioned other areas of complexity for RISCs: multiple functional units, instruction scheduling, and branch avoidance. A partial list of other areas includes:

- recognition of procedure leaves and elimination of prolog/epilog sequences;
- local stack-frame organization, or where to place locals, the static link (for Pascal up-level addressing), the register spill areas, the outgoing argument-overflow areas, and register arguments whose addresses were taken;
- optimization of memory-block-move and string-compare operations;
- the division or multiplication of integers when machines do not provide single instructions (most CISCs provide such instructions and also optimize for special cases of small integers);
- memory load/store latency; and
- cache performance.

Note that local stack-frame organization is especially complex due to the limited displacement range that is typical in RISC base-displacement addressing. While a few of these concerns are relevant to CISCs as well, unlimited-displacement addressing typically costs more in RISCs, which amplifies the importance of stack-frame organization in this type of processor. Furthermore, each specific RISC chip has its own set of quirks that demands careful attention. One example is the extremely fast, pipelined, dual-instruction mode of the i860.

The moral of the story is that as manufacturers try to drive performance to even higher levels, the list of challenges in RISC compiler code generation will continue to grow.

References

1. D.A. Patterson and C.H. Sequin, "A VLSI RISC," *Computer*, Vol. 15, No. 9, Oct. 1982, pp. 8-21.

2. Pyramid Technology Advertisement, *Computer*, Vol. 17, No. 10, Oct. 1984, p. 11.
3. *Draft Proposed American National Standard for Information Systems—Programming Language C, Doc. No. X3J11/88-159*, Computer and Business Equipment Manufacturers Assoc., Washington, DC, Dec. 1988.
4. *i860 64-bit Microprocessor Programmer's Reference Manual*, No. 240329-001, Intel Corp., Santa Clara, Calif., 1989.
5. *Intel i860 Architecture UNIX System V/860 Binary Standard*, Intel Corp., 1989.

Thomas J. Pennello is vice president and cofounder of MetaWare Incorporated, where he develops and oversees the development of compilers. His primary interests include compilers, translator-writing systems, and language design.

Pennello received a BS degree in mathematics from the University of Santa Clara and the MS and PhD degrees from the University of California, Santa Cruz. He is a member of ACM, the IEEE Computer Society, and ANSI's X3J11 Technical Committee for standardizing the C programming language.

Direct questions regarding this article to the author at MetaWare Incorporated, 2161 Delaware Ave., Santa Cruz, CA 95060, or uunet address tom@metaware.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159 **Medium** 160 **High** 161

If you liked this issue, stay tuned for June!
Read *IEEE Micro* for more...

Articles from the Computer Society TCMM's Hot Chips Symposium:

- Motorola's 68040, part 2
- Texas Instruments' Floating-Point Unit
- Endian Wars: The Negotiations Continue
- 88000 Family Update
- 8087 Family Stack Problems
- 32-bit MMU Design Comparison

All this and much, much more!

IEEE MICRO

FEATURE

Developing the GX Graphics Accelerator Architecture

High-level graphics on entry-level workstations become practical with this new approach to acceleration.

Curtis R. Priem

Sun Microsystems, Inc.

Sun's GX Graphics Accelerator Board (shown in Figure 1 on the next page) represents a significant departure from traditional graphics acceleration. Previous graphics accelerators usually contained some intelligence, either a CPU or bit-slice processor, which controlled the specialized graphics hardware and processed much of the rendering calculations. The support hardware, memory, cache, power supply, and board size prevented, until now, graphics acceleration from being placed on entry-level workstations. In the GX, however, the host CPU functions as the intelligent controller and two large ASICs (application-specific ICs) supply hardwired graphics functions.

The 80/20 rule and graphics

We determined the GX's functionality based on the dollar size of its target markets. We applied a variant of the 80/20 rule popularized in RISC (reduced instruction-set computer) designs to graphics algorithms. This method identifies the 20 percent of CISC (complex instruction-set computer) instructions that are used 80 percent of the time. These become the instructions for the RISC microprocessors. The 80/20 rule, as it applies to the GX, meant identifying the 20 percent of the graphics algorithms that would have the biggest impact on at least 80 percent of the dollars within each targeted market sector.

0272-1732/90/0200-0044$01.00 © 1990 IEEE

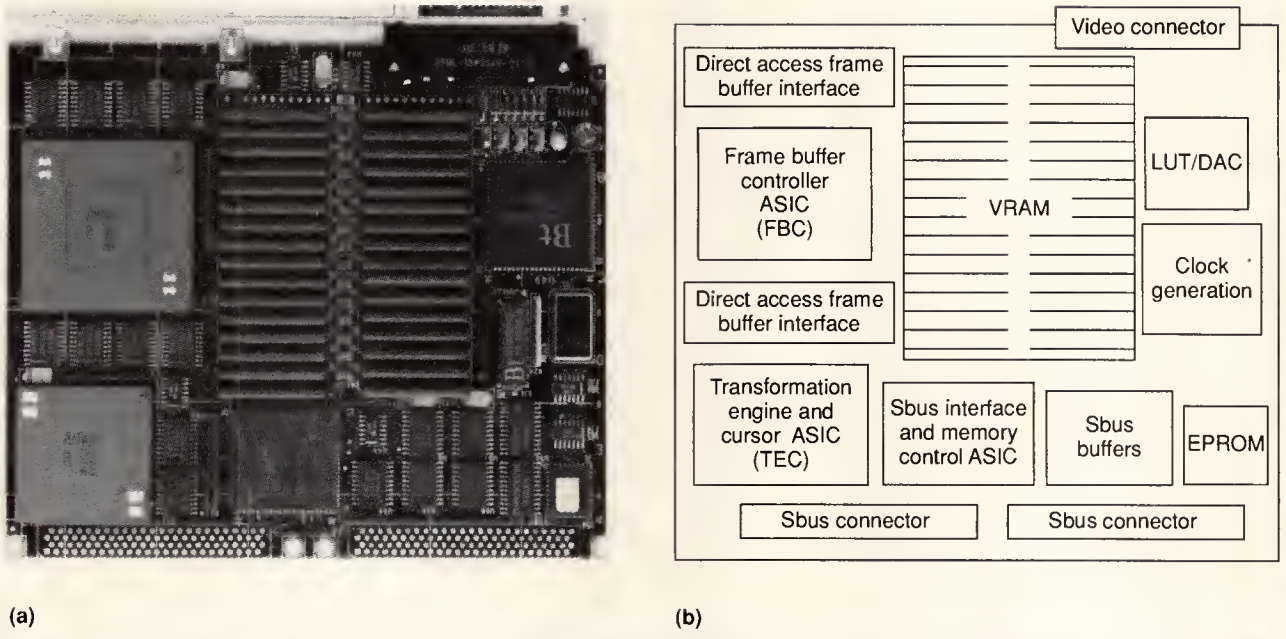


Figure 1. The GX Sbus Graphics Accelerator Board (a) and the layout diagram (b).

An arbitrary quadrilateral is the GX's only geometric primitive. The GX directly supports the most commonly used graphic drawing primitives (points, lines, triangles, rectangles) as degenerate cases of a quadrilateral. The operations and primitives that the GX does not directly support are generally available through multiple applications of one (or more) of the supported operations. This approach resembles the way CISC instructions can be made up by combining simpler RISC instructions.

The GX does not support circle and arc primitives. However, it can readily approximate them with short line segments. In addition, the GX supports flat shading of images only when every pixel on a polygon is the same color or intensity. However, by breaking the object into many smaller objects (tessellation) each with its own color, users can obtain a visually acceptable approximation to Gouraud-shading techniques.[1,2] The high performance of the GX's rendering hardware makes this approach attractive. Going one step further, when a tessellated polygon gets close to one pixel in size, the result becomes the Phong-shading technique.

The GX accepts and entirely completes its low-level graphics operations without host assistance. However, the GX does not control nor drive the graphics application; the host CPU continues these functions. The host executes the application until it encounters a function supported by the GX (directly or indirectly by compositing multiple supported operations). At this point, the CPU writes parameters of the function to the GX's memory-mapped data registers. The GX first validates these parameters prior to the start of any operation. The CPU reads a status register to see if any unsupported or errant condition exists. If one does,[3] the host CPU handles the exception in software.

Integration

High-level integration was one of the key ideas behind the GX concept. In fact, the code name for the GX during its development was LEGO, which stood for Low End Graphics Option. "Low end" did not describe performance; it described the targeted workstations. The GX board (including frame buffer) requires less than 20 watts from the workstation's power supply.

In previous graphics accelerators the performance was proportional to its price. With large graphics accelerators the user must pay for a large power supply, extra cooling, and a large chassis. These added costs alone have kept high-performance graphics from appearing on desktop computers.

To achieve GX suitability for the low-end workstation, we chose to integrate the GX into two large ASICs,[4] the Frame Buffer Controller and the Transformation Engine and Cursor chip. The FBC acts as the graphics-rendering engine. It contains 43,000 gates (170,000 transistors) packaged in a 223-pin grid array and uses LSI Logic's 1.5-μm sea-of-gates technology. The TEC transforms all the graphics, provides a hardware cursor, and contains the timing generation logic for the frame buffer. It contains 25,500 gates, over

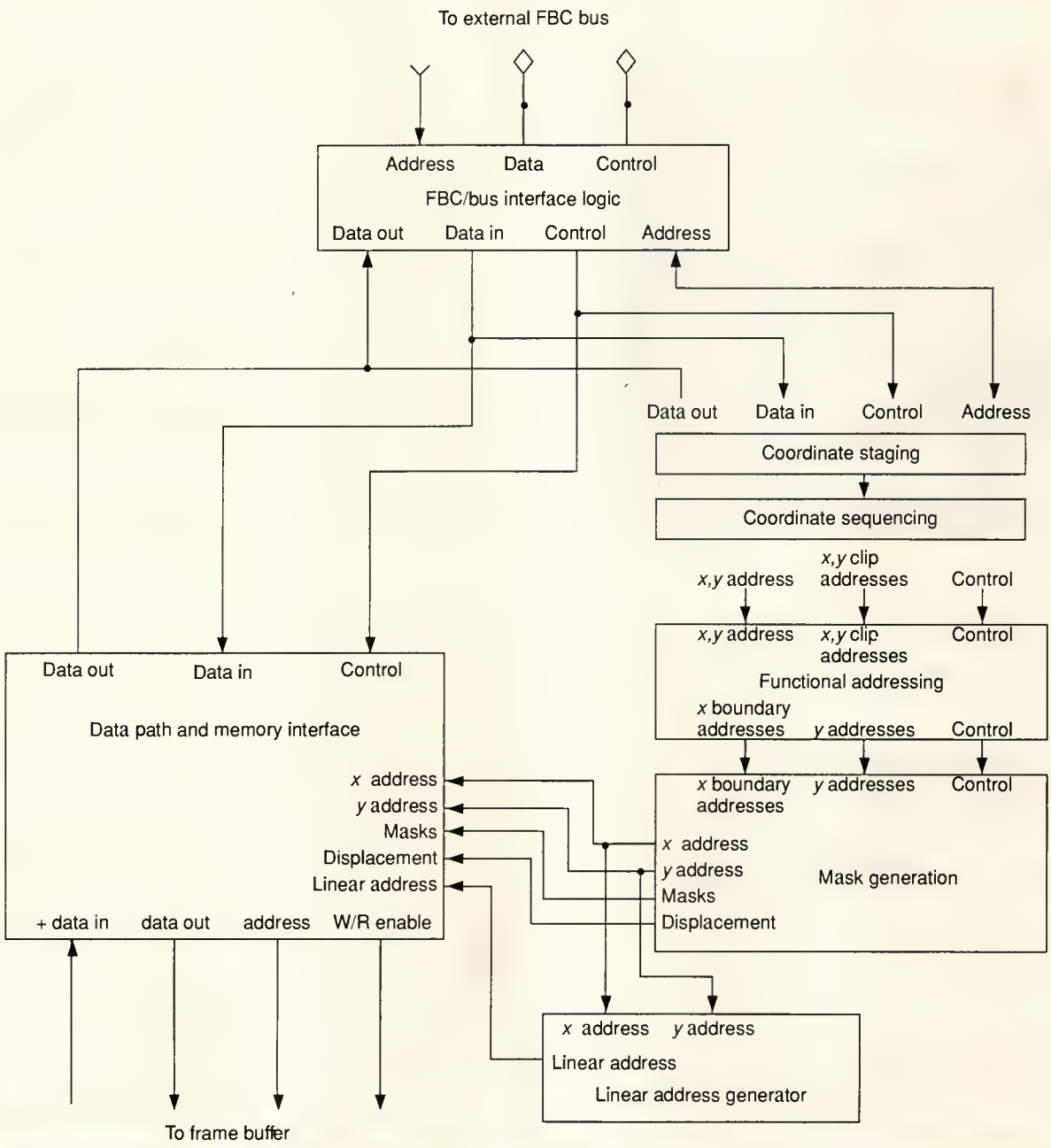


Figure 2. FBC block diagram.

4 Kbits of RAM (212,000 transistors) in a 95-PGA package, and uses LSI Logic's 1.5-μm standard-cell technology.

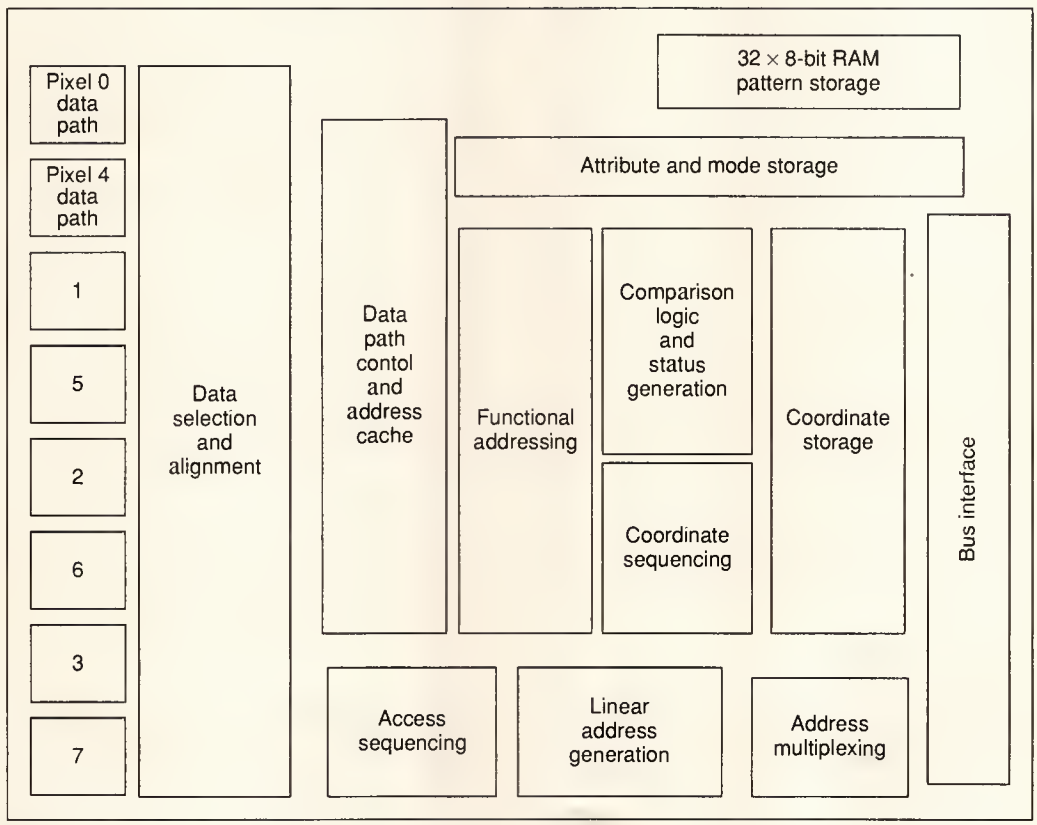
Frame Buffer Controller chip

The FBC chip detailed in Figures 2 and 3 (on the next page) performs all rendering or drawing operations and places the results in the frame buffer. It processes only three instructions:

- DRAWs, which render arbitrary, filled quadrilaterals,
- FONTs, which display precomputed pixel images, and
- BLITs, which transfer block images.

The first selected rendering algorithm draws only arbitrary, filled quadrilaterals that can be self-intersecting and degenerate. Due to limited silicon resources, we felt it very important to simplify geometric rendering into one algorithm in the FBC.

(a)



(b)

Figure 3. The FBC die (a) and its layout (b). (Die photomicrograph courtesy of LSI Logic.)

Since it processes degenerate quadrilaterals (repeated vertices), the FBC can render points, lines, and triangles.

To define each quadrilateral to be drawn, four (x,y) coordinate pairs must first be loaded into the FBC. Special addressing modes allow a point, line, triangle, or rectangle to be completely defined by loading the minimal number of vertices. The chip can express a point as a degenerate quadrilateral in which all of the vertices are the same. In this case, the chip's addressing

logic replicates the one vertex into all four vertex registers. In a similar manner, the programmer can specify a line with just two vertices, a triangle with three, and an aligned rectangle with two opposite corner vertices. The FBC also supports chained addressing, which avoids reentering common vertices in polylines, triangle/quadrilateral strips, and meshes.

The FBC renders the object defined in its registers using a variant of the Bresenham line-drawing algorithm.[5-8] Internal to the FBC, two Bresenham algorithm engines run in parallel, each processing one edge of the arbitrary quadrilateral. Additional logic identifies the left and right edges for each scan line, sorts them, and applies clipping calculations. The scan line then fills between these two points with data from a pattern RAM. Degenerate quadrilaterals, such as points or lines, simply cause the two Bresenham engines to process the same edge. This duplication is of no consequence since the two engines run in parallel (no time lost), and the sorting logic can still identify the proper scan-line boundaries.

The drawing operation renders only into a specified rectangular region referred to as the clip window. Processed, invisible (clipped) pixels do not represent productive work in that no change to the frame buffer results. For most clipped objects, standard graphics software would first compute and then render a new object that represents the geometric intersection of the clip window and the original object. The computation of the vertices of the new object involves time-consuming division operations. Programmers can make a performance trade-off between deferring clipped object processing to the host or allowing the accelerator to handle it. In supporting this trade-off, the FBC architecture defines a clip test window that surrounds the specified clip window. Users can set this test window at a programmable distance in both x (edges) and y (horizontal lines) coordinates beyond the edges of the clip window.

The FBC completely and efficiently processes a clipped object if its vertices are all contained within the clip test window (which is generally larger than the clip window). The host software more efficiently (in terms of time) renders objects that exceed the test window's boundaries. Since the size of the test window is programmable, users should set it to optimize each host workstation.

The FBC will actually handle many objects that have vertices outside the test window when certain conditions are met. The flexible drawing-engine architecture allows an object to be rendered in either top-to-bottom or bottom-to-top order. The drawing engine renders an object whose left, right, and bottommost vertices appear inside the test window, with the uppermost vertex outside it, from bottom to top. It stops when it reaches the upper edge of the clip window. Thus, the pixels at the top of the object that exceed the clip window need never be processed.

The FBC also accelerates the rendering of precomputed raster images into the frame buffer. These images can exist in either a monochrome (1-bit-per-pixel) format (text) or a color (8-bit-per-pixel) format (image). The FBC accelerates the display of text and images by supporting x/y addressing, address autoincrementing, clipping, raster operations, and frame-buffer alignment. The FBC also provides 1-to-8 plane expansion for monochrome text or images.

The FBC determines the target location of a font by loading the horizontal left and right points of the scan line in which the font will be placed. This arrangement allows different-width fonts (in terms of pixels) to be written. By specifying the font width this way, versus providing a pixel-width value, the same clipping and address processing logic that handles quadrilaterals handles fonts. (The operation is similar to drawing textured horizontal lines.)

Once the font widths are specified, users can automatically increment the font's x and y addresses between the receipt of each font data access by any amount. This capability reduces the amount of information that the FBC must have for consecutive font accesses to just the data itself.

The BLIT command to transfer block images permits one rectangular area of the frame buffer to be copied to another area of the same size. The FBC specifies source and destination areas for BLITs by loading the diagonal corners of the source and destination rectangular regions into the vertex registers. As with any type of memory-to-memory copy operation, care must be taken when the source and destination elements of the copy overlap. Overlapping elements dictate how the copy operation must progress to ensure data integrity. For BLITs, this means the copy may be constrained to occur in left-to-right, right-to-left, top-to-bottom, or bottom-to-top operations. The FBC determines and executes a BLIT for arbitrary overlaps in the required progression without intervention from the host.

FBC graphics attributes

The FBC supports a rich set of attributes that define where and how new pixel data merges into the existing pixel data in the frame buffer. This approach is analogous to a CPU that supports a generic operate instruction in which the actual ALU operation to be performed is specified in some auxiliary register.

The architecture stipulates that all attributes are available for use and consistently applied across all of its three graphics commands. An example of a common attribute is color; the FBC supports both foreground and background color. Once established, the colors and other attributes will affect all subsequent operations.

The predominant mechanism for defining the way a new per-pixel value merges bit-wise with the existing value involves raster operations (Rops). The FBC's

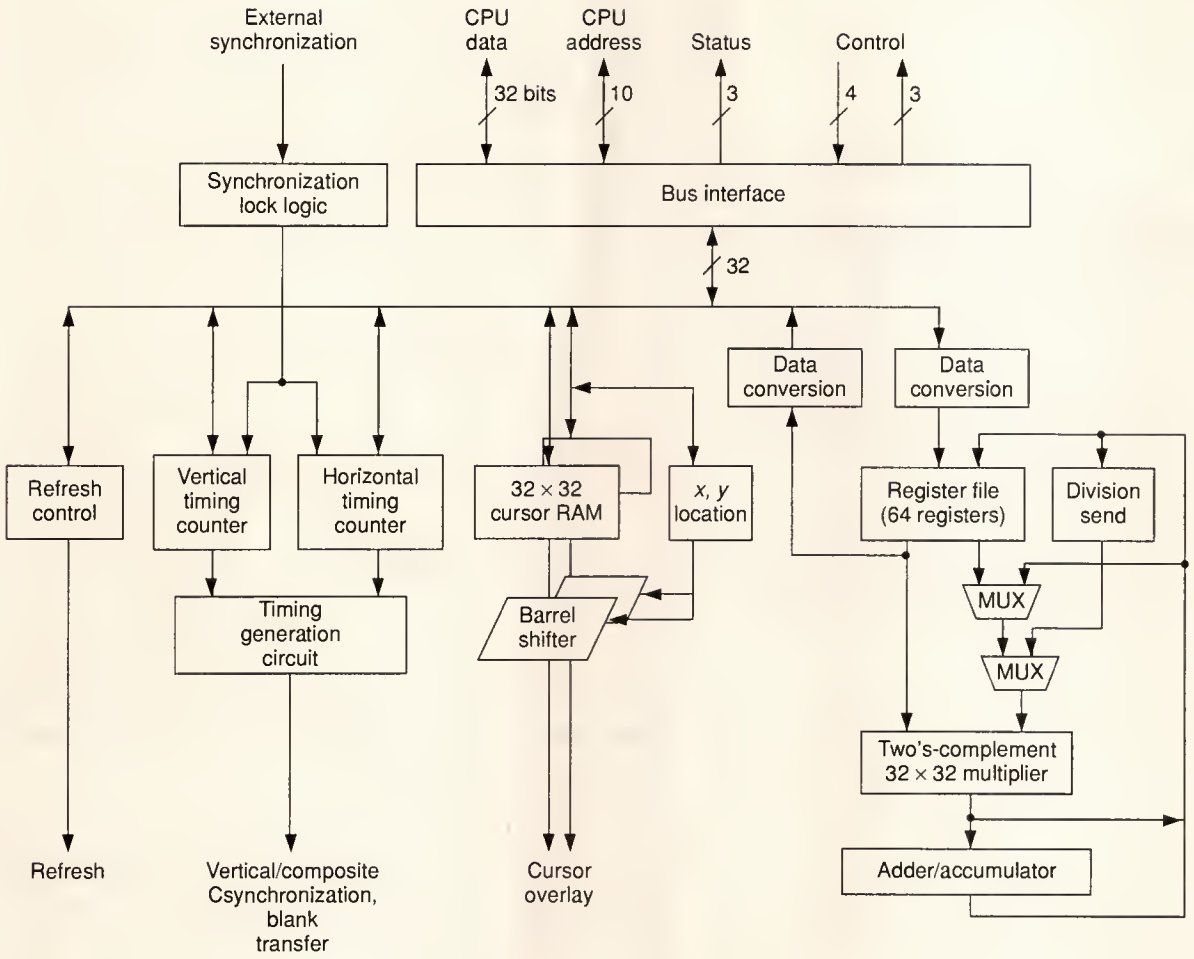


Figure 4. TEC block diagram.

Boolean Rop mode performs logical (And, Or, NAND, XOR, SRC, DEST) bit-wise operations between each new source pixel and the existing destination pixel at the target address of the frame buffer. The FBC supports all 16 Boolean Rops as defined in Sun's Pixrect graphics library. In addition, the FBC integrates the foreground and background colors into the final Rop determination, permitting 65,536 possible Boolean Rops.

A plane-masking capability lets users select the bit planes within each pixel for modification. The current implementation of the FBC uses the write-per-bit capability of its video RAMs to restrict individual planes from being modified. Thus, each VRAM actually performs the plane-masking calculations so that no performance degradation occurs when users enable this option.

The FBC uses a pattern RAM activated by the Draw command. The 16 × 16, one-bit-deep (monochrome) pattern fills quadrilaterals. The pattern repeats across the frame buffer every 16 pixels and can be aligned to arbitrary offsets in both x and y directions. The FBC's color and Rop capabilities permit the monochrome pattern to expand into an 8-bit color index.

The TEC chip

The transformation engine and cursor chip modifies all graphics, supports the computational requirements of graphics programs,[2] provides a hardware cursor, and contains the timing generation logic for the frame buffer. (See Figures 4 and 5.) The TEC contains a 64-register file and a 32-bit, signed multiply/accumulator, which combine to provide arithmetic calculations at a rate of up to 50 million floating-point operations per second.

The TEC includes explicit support for the typical 2D and 3D graphics-transformation pipeline. The

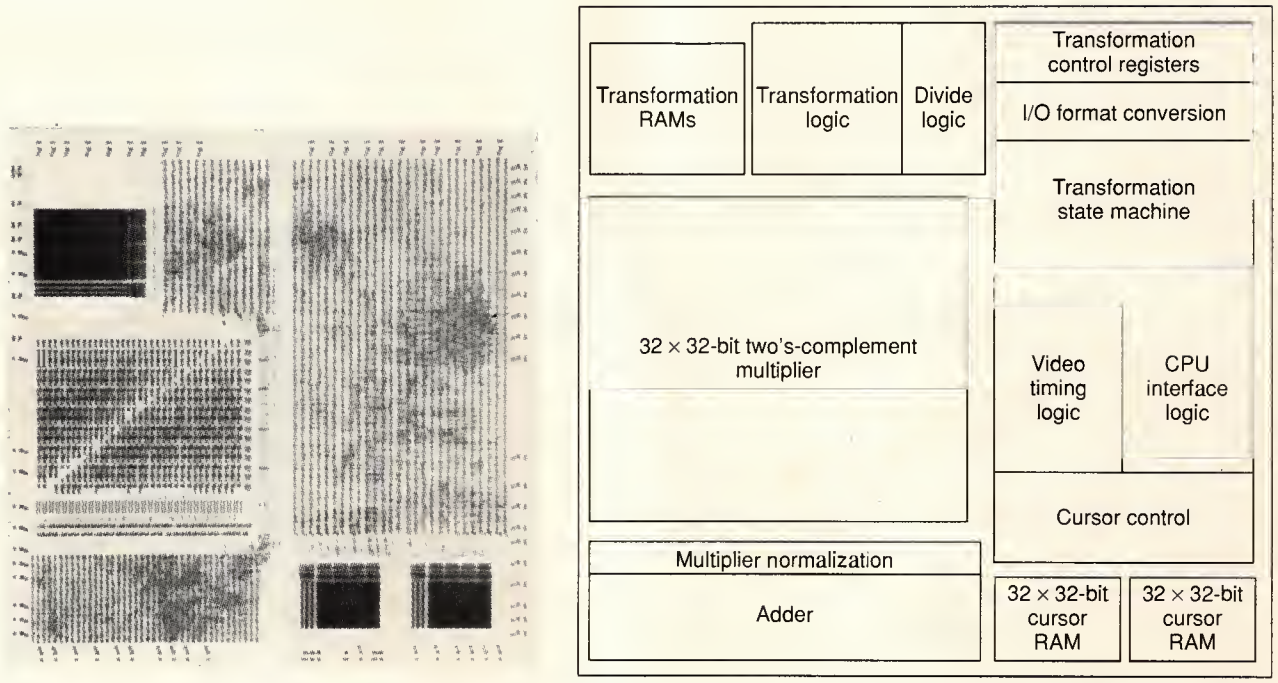


Figure 5. The TEC die (a) and its layout (b). (Die photomicrograph courtesy of LSI Logic.)

modeling and viewing, or MV, matrix can consist of anything from a simple 3 × 2 matrix used by 2D graphics to a 4 × 4 matrix used by 3D graphics. The MV matrix accommodates rotation, scaling, and translation of an object or its viewing position. An optional clip-checking operation adjusts a rectangular region for 2D graphics, a viewing pyramid for 3D graphics, and a double pyramid for homogeneous graphics such as parametric curves.[9] The TEC also provides an optional perspective divide function that uses Newton-Raphson iterations.[10,11]

At the end of the pipeline, an optional matrix (the virtual device coordinate) transforms the coordinates into the final screen-coordinate space. The host can read and/or automatically load transformation results into the FBC for rendering. The host also has access to intermediate pipeline results.

The TEC's external interface supports three different data formats. Data can be read or written in either signed integer, fract (16.16), or IEEE-754 single-precision floating-point formats.[12] Different formats can be intermixed, and all conversions are pipelined to avoid a performance penalty. The TEC accepts both absolute and relative 2D (x,y), 3D (x,y,z), and homogeneous (x,y,z,w) coordinate data. Since the TEC's math capability extends to general-purpose vector operations, it calculates the arithmetic required for matrix concatenation, dot products, lighting models, and even some common imaging operations such as convolutions.

The TEC provides a three-color (and transparent), 32 × 32-pixel hardware cursor. We included this capability when we realized that support of a software-based cursor uses as much as 20 percent of the host CPU's time. So, with a workstation processing 5 million instructions per second, the CPU can devote as much as 1 MIPS to manipulating the cursor and its bitmap. With the TEC's hardware cursor, the CPU loads only a new (x,y) cursor coordinate into the hardware when the mouse moves. The three-color cursor accommodates images with an outline in the opposite color of its body. This capability prevents the cursor from ever being lost in a solid-colored background (as in a window border).

The TEC also contains the timing generators. A programmable timing generation circuit allows software to program the TEC for different-resolution monitors and different frequencies. The TEC generates the signals required for the VRAM's transfer and refresh cycles, and the synchronization and blanking signals required by the monitors.

Software in silicon

Even though the GX interface consists of only three simple instructions and appears very much like a RISC, internally the ASICs function in "super-CISC" style. The chips don't have the complex instructions found in CISC processors; rather they perform the equivalent of very complicated software subroutines.

The GX provides the functional equivalents of

10,000 lines of code. Neither the FBC nor the TEC ASICs contains actual software or microcode. The subroutines proceed instead in a massively parallel way, via state machines (the FBC contains 12 state machines) and combinatorial logic (the FBC's Rop logic and barrel shifters repeat eight times). Since we lost most resemblance to the original routines, we developed numerous new algorithms and have applied for 15 patents related to the FBC and six patents associated with the TEC.

This design approach meant ignoring the normal sequential way of implementing a subroutine or algorithm. For the GX, we used large amounts of combinatorial logic operating in parallel with many state machines that supported wide data paths. (The FBC's main data path is 128 bits wide.) We emphasized a steady, high data flow rate and did not limit the number of gates. We also provided numerous replicated resources just to enhance overall throughput.

Infinitely fast CPUs

As mentioned earlier, the most common way of designing a graphics system is to build a board that has some type of intelligent device on it. The device may be a microprocessor or a microcoded, custom bit-slice processor. Unfortunately, this device uses the same basic technology as the host CPU but is put together in a little different way to optimize the graphics function. When the next-generation technology comes along, the host CPU generally performs faster, quickly making the current-generation's graphics subsystem obsolete.

A good example of this phenomenon relates to the development of the original IBM Professional Graphics Adapter. PGA designers decided to use an 8088 processor to handle image-rendering operations for the frame buffer. This design worked well on the company's PC XT, which also had an 8088, and even on its AT, which had an 80286 processor. In these two cases, the multiprocessing arrangement was effective.

But upon adding the original PGA to a 80386-based PC, the overall system performance actually slowed down. The 80386 can both execute the application software and render the graphics in less time than the PGA could render just the graphics—not an effective multiprocessing setup. As things turned out, the 80386-based PCs were available within two years of the time the PGAs became available.

The rate of increase in CPU performance seems to be increasing with the advent of the new RISC CPUs. Following company cofounder Bill Joy's Law, we can expect CPU performance to double roughly every year [MIPS = 2 raised to (current year − 1984)]. In other words, designers would have to develop a new graphics board every year.

Instead, we wanted to come up with an architecture that would last for several years. We knew that meant that the GX graphics accelerator could not become the bottleneck of the system if a 100-MIPS CPU was connected to it. The GX could not perform slower than the worst case in which an infinitely fast CPU was connected directly to the frame buffer.

With such a CPU connected to a frame buffer, the VRAMs become the bottleneck of the system. We knew that if we built our graphics accelerator, the worst possible thing that could happen would be that 10 years from now the CPU would render as fast as the accelerator.

The bottleneck would occur at a different location in our design. A common bottleneck for previous accelerators had been determined by the transformation speed that could be achieved using the chosen technology. Designers then developed the rendering section to draw polygons only at the rate that transformed coordinates could be delivered.

Table 1 (on next page) lists the device and related frame-buffer bandwidths for a number of common VRAM organizations. We assumed a one-megapixel, or 1,024 × 1,024 × 8, frame buffer. Notice that even though the bandwidth of each VRAM generally increases, the time required to load all new data into the VRAM lengthens, and the bandwidth into the frame buffer decreases. This phenomenon is currently unique to graphics operations. In these operations the size of the frame buffers does not increase with time due mainly to the cost of higher resolution monitors. Workstations may start to run into this problem in a few years unless the operating systems, windowing systems, and applications continue to increase in size proportionally to dynamic-RAM density. Since the interface to the VRAM is not going to double in performance every year, we achieve a very stable interface for designs.

Scalable performance

The GX saturates the bandwidth of the currently available VRAM. Our design goal—having any drawing operation continuously access the VRAMs at their maximum bandwidth—meant that we had to design each of the three algorithms in the GX from the frame buffer backwards to the CPU. Let's look at what happens when the GX draws a polygon.

The FBC horizontal-fill hardware must process a horizontal span at the rate the VRAMs can accept data. The current board allows a page-mode read or write access to 16 pixels in the VRAM every 129 nanoseconds (or in three 43-ns FBC cycles). The 9-ns difference between this cycle time and the theoretically fastest cycle time of the VRAM (120 ns) allows a margin for manufacturing reliability. Read-modify-write cycles (required by raster operations) take seven cycles (301 ns) for 16 pixels. The cycles occur in the FBC's data path and memory interface

Table 1.
VRAM device and related frame-buffer bandwidths.

VRAM size (bytes)	VRAM configuration (bits)	Page-mode cycle time (ns)	Time required to load each VRAM (ms)	Bandwidth of each VRAM (Mbytes/s)	Number of VRAMs	Total bandwidth into frame buffer (Mbytes/s)
256K	64K × 4	120	8	4	32	128
1M	128K × 8	60	8	16	8	128
1M	256K × 4	60	16	8	8	64
4M	512K × 8	60 *	32	16	2	32
4M	1M × 4	60 *	64	8	2	16

* We've heard talk of implementing a superfast page-mode cycle (25 ns) in future VRAMs.

section shown in Figure 2.

The mask-generation logic must calculate the intersection of the clip window and the edges of the polygon to determine which pixels in the frame buffer are changed at the rate that the memory interface section can access the VRAM. In this case, the worst case object would be a vertical line. Since the horizontal-fill logic can write data every three cycles (129 ns), the mask-generation logic must create a write-enable mask every three cycles.

The functional addressing logic contains the two Bresenham state machines, which must calculate the two x coordinates of the polygon for a given y coordinate. It also contains the clip boundary information. It must calculate and "walk" the polygon's edges at the rate that the mask-generation logic can accept data.

The coordinate-sequencing stage decides which pairs of edges of the polygon are currently being drawn. This bottleneck in firmware-driven graphics accelerators occurs because the decision process proceeds serially. In the FBC combinational logic, which looks at information from the coordinate-staging logic, makes the decision. The coordinate-staging logic creates setup status information based on the vertices of the polygon. The FBC sets up the information in 86 ns (two clock cycles) by comparing the four vertices 56 times with each other and against the clip and clip test windows. Thus the FBC generates 44 bits of comparison information, which is then available for immediate use. Determining whether the FBC must render with a self-intersecting, hidden, or degenerate polygon becomes a simple combinational decoding operation.

The CPU could saturate the FBC if it could read data from main memory and write it to the FBC every 86 ns. The GX's current hosts take between 350 ns and 500 ns to complete read and write cycles for the accelerator. This speed leaves plenty of room for increased performance when the GX runs in faster, future systems.

The CPU has the option of writing the coordinates first to the TEC where they can be transformed and then automatically loaded into the FBC. To avoid becoming the bottleneck, the TEC would have to transform the next set of coordinates in the time it takes to draw an average-size object on the screen (a 10-pixel line or 100-pixel quadrilateral). We've determined the floating-point processing speed requirement to be about 50 Mflops.[13]

When the functionality and performance of all of these logic blocks are put together, the CPU becomes the bottleneck of the graphics system. The GX's performance is directly proportional to the rate the CPU can write data into the GX. We've marketed the GX as being scalable to the CPU's MIPS performance, but it is really scalable to the workstation's I/O performance. Although in general I/O performance has been scaling along with the CPU MIPS and Mflops rates, it is not always in the same proportions. Thus, we can expect GX performance to increase in the future, though it will vary from system to system.

If the GX already runs at the bandwidth of the VRAMs, where will we find the next leap in performance? One improvement could result from the workstation and computer manufacturers' assuming a very active roll in the design of future VRAMs to make sure bandwidth improvements occur. Care must be taken in adding fancy features to VRAMs such as raster operations or filling modes that require information to be stored in the VRAM. In multitasking systems the operating system kernel would have to "understand" how to context switch this information from the VRAM. Other areas for overall system performance improvements could result from moving to the GX those graphics algorithms presently processed in software by the CPU.

Due to the GX design approach, next-generation

Table 2.
Comparing performance in two systems.

Parameters	Sun Microsystems Sparcstation 1GX 4/60GX-8	Silicon Graphics Personal Iris 4D/20	Units
2D vectors	470	80	Kvectors/s
3D vectors	244	80	Kvectors/s
Three-sided, flat-shaded polygon (mesh 50 pixel)	244	16	Ktriangles/s
Four-sided, flat-shaded polygon (independent 10 × 10 pixel)	10	10	Kquadrilaterals
Screen clearance time	8	8	Milliseconds

ASIC techniques can offer little in the area of performance improvements. Adjusting ASICs to higher frequency clocks doesn't mean that the VRAMs can be clocked any faster. The only area to benefit from faster silicon is the transformation engine. Even the TEC's 50-Mflops processing rate is not enough to sustain the rendering hardware at the VRAM bandwidth limits when the GX runs with faster CPUs. The TEC will need to increase proportionally with the rate the CPU can load data into the GX. Beyond this, however, the new submicrometer CMOS processes offer only cost reduction and further integration opportunities.

Comparison

The design approach and concepts embedded in the GX place it in a price/performance class by itself. Table 2 compares performance rates in the Sparcstation 1GX and Silicon Graphic's Personal Iris.[14] Both companies comparably configured these systems at about $15,000.

The 3D vectors and the three-sided polygon meshes both require only one coordinate to be loaded per item drawn. Since the GX uses the same algorithm to draw both a vector and a triangle, the performance rates between these benchmarks are the same. The 3D independent, four-sided polygons require four times the amount of data to be loaded per item than do vectors or triangle meshes. The quadrilaterals are not exactly four times slower than the vectors because the status of the object that is being drawn must be read from the FBC for each object drawn. The status tells the CPU if any exception has occurred such as numerical overflows or full buffers. If no exception is present, the act of reading the FBC initiates the drawing operation. Also, notice that both systems can clear the frame buffer in the same amount of time, which is the same as the bandwidth of the currently available VRAMs (64K × 4 and 128K × 8).

Lowest common denominator

We wanted the GX to bring Sun one step closer to redefining the base functionality of a workstation. Realization of this goal could include the GX in the lowest common denominator (meaning nonoptional, always available on all platforms) of capability between different models of workstations. At first, this may not seem that important a position to attain, but from an application programmer's point of view, the base configuration becomes very important.

Always having graphics acceleration available means that an application can be designed to use display lists and other techniques to exploit the acceleration. An application can be highly tuned if it is to run on only one workstation configuration. If it needs to run on an unaccelerated frame buffer, designers would have to compromise on the features and performance between optimizing for the accelerated and the unaccelerated case. Math-intensive applications evolved the same way, until the floating-point units became standard even on the most cost-sensitive workstations.

With the high levels of integration and the minimal part count of the GX (the entire Sbus-based GX fits on a 6 × 5-inch PC board), somewhat inevitably we will soon see the cost difference between an unaccelerated frame buffer and the GX be reduced to insignificant levels. At that point, the GX could become the graphics workstation's least common denominator.

Software interface

Previous graphics systems containing some type of controller normally communicated data through a display list. This list is simply a well-defined data structure of graphics commands and the coordinates of the objects that the graphics system is to draw. The host CPU would normally create the display list by moving

the data from the user's or application's database into the display list data structure. This data would then be fed to the graphics system through shared memory and first-in, first-out or direct memory access. The graphics system would next transverse the display list and render the object. Users had to buy a dual-processor system to move data around twice as inefficiently.

However, with the GX the CPU becomes an intelligent DMA controller. The CPU reads the data out of the graphics language's display list and writes it into the memory-mapped registers in the GX's ASICs. This design makes the GX very flexible since every graphics language (PHIGS, GKS, or Pixrect) has its own display list formats.

The CPU also makes meaningful decisions part way through the rendering of a picture. For example, if an object moves too far away from the viewer, the CPU could decide against drawing the object and skip to the next one.

GX programmers work in a native environment with high-level languages such as C and debugging tools like dbx on their own workstations. Special simulators, assemblers, microcode, or hardware are unnecessary since the programmers are using the same processor for the graphics language as they are using for driving the GX. High-level graphics programmers still control low-level GX interactions by accessing any and all GX registers. This level of device control had previously only been available to the firmware microcoder on most other graphics subsystems.

The GX accelerates more than vectors and polygons; the most used graphics program is the windowing system itself. The GX provides acceleration for Sun Windows, Sun Tools, X11/News, and Open Look. Users can accelerate applications graphics after porting to one of the graphics interfaces we support, such as Pixrect, GKS, or PHIGS. The GX contains an unaccelerated, dumb frame-buffer interface so that any application that does not run on one of these graphics languages will still work, just slower.

References

1. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley Publishers, Reading, Mass., 1982.
2. W.M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill Book Company, New York, 1979.
3. C.A. Malachowsky and C.R. Priem, "The GX Graphics Accelerator Architecture," *Sun Tech. J.,* Sun Microsystems, Inc., Mountain View, Calif., Vol. 2, No. 4, Autumn 1989, pp. 85-93.
4. C.A. Malachowsky, "Designing a Low-Cost Scalable Graphics Accelerator," *High-Performance Systems,* Vol. X, No. 5, May 1989, pp. 68-78.
5. J.E. Bresenham, "An Incremental Algorithm for Digital Plotting," *Proc. ACM Nat'l Conf.,* Aug. 1963.
6. J.E. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM System J.,* Journal 4, No. 1, July 1965, pp. 25-30.
7. J.E. Bresenham, "Run Length Slices for Incremental Lines," *IBM Technical Disclosure Bulletin 22-8B,* Jan. 1980, pp. 3744-3747.
8. J.E. Bresenham, "Incremental Line Compaction," *The Computer J.,* Vol. 25, No. I, 1982, pp. 116-120.
9. J.F. Blinn and M.E. Newell, "Clipping Using Homogeneous Coordinates," *ACM Computer Graphics,* Vol. 12, No. 3, Aug. 1978, pp. 197-203.
10. J.J.F. Cavanagh, "Digital Computer Arithmetic: Design and Implementation," McGraw-Hill, 1984, pp. 278-284.
11. S. Waser and M.J. Flynn, *Introduction to Arithmetic for Digital Systems Designers,* CBS College Publishing, New York, 1982, pp. 196-199.
12. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic,* IEEE Computer Society Press, Los Alamitos, Calif., 1985.
13. K. Akeley and T. Jermoluk, "High-Performance Polygon Rendering," *ACM Computer Graphics,* Vol. 22, No. 4, Aug. 1988, pp. 239-246.
14. Silicon Graphics Inc., *Personal IRIS Specification, (Preliminary),* Mountain View, Calif., Oct. 1988, p. 4.

Curtis R. Priem, a staff engineer at Sun Microsystems, Inc., is the architect of the GX graphics accelerator products. He also designed the transformation engine section of the GX's TEC. Prior to joining Sun, he worked at Genrad, where he developed the Hichip hardware modeler. He also was the architect and principal designer of IBM's Professional Graphics Adapter while at Vermont Microsystems Inc.

Priem holds a BS degree in computer systems engineering from Rensselaer Polytechnic Institute.

Direct questions concerning this article to the author at Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, or by electronic mail at curtis@sun.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 168 **Medium** 169 **High** 170

Developing the WTL3170/3171 Sparc Floating-Point Coprocessors

The WTL3170 and WTL3171 floating-point coprocessors are the first in a family of Sparc-architecture-compatible parts from Weitek. With varying frequency and latency performance, the chips work with the first two integer unit (IU) implementations from other Sparc vendors. These are the first Sparc-architecture chips to integrate all floating-point controller functions; floating-point register file; and 64-bit ALU, multiplier, and divide/square-root units in one die. A strong OEM relationship in system behavioral-level modeling and a short time to production were key factors in our product development plan.

Contending with dual floating-point interfaces at both 25 and 40 MHz posed an extraordinary challenge in coprocessor development.

Mark Birman
Allen Samuels
George Chu
Ting Chuk
Larry Hu
John McLeod
John Barnes

Weitek Corporation

Project development goals

Our objective was to produce a single device that would 1) using the 3170 mode, operate with the 25-MHz Fujitsu S-25 processor,[1] and 2) using the 3171 mode, also function with the 40-MHz Cypress 601[2] integer units. We wanted to achieve this functionality with only a process feature upgrade from 1.2- to 1.0-μm.

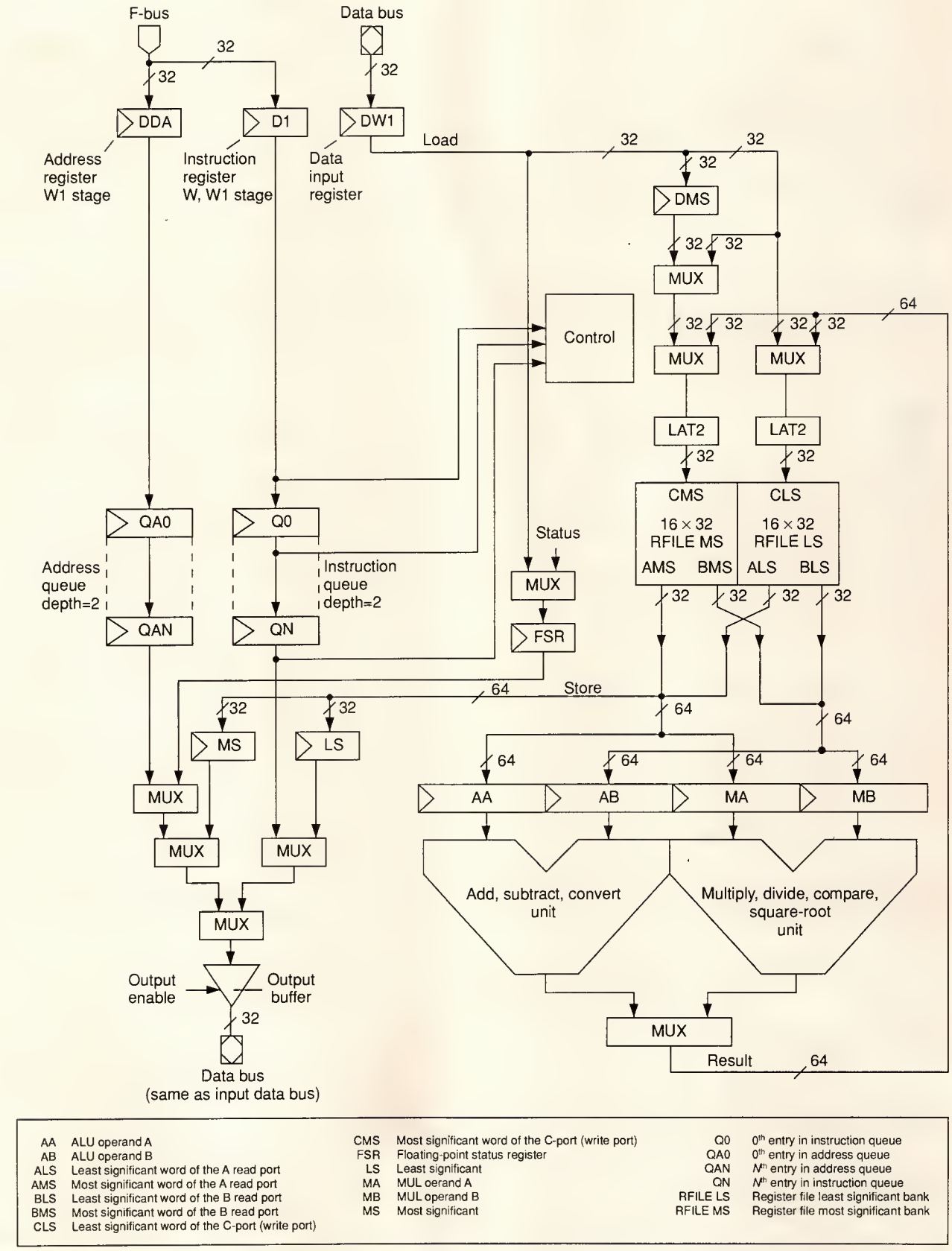
Our technical goal, therefore, was to combine a 64-bit floating-point core, floating-point controller, and instruction queue modules. Because we began architecture development in April 1988, and had to be production-qualified by March 1989, we chose the floating-point core unit from our 3×64 product[3] and began functional modeling of the controller requirements.

Contending with two floating-point controller interfaces was, of course, our major challenge. However, because of the schedule for the forthcoming Sparcstation 1 from Sun Microsystems, developing the S-25-compatible component took first priority.

Implementation goals

Functional correctness and full compliance with the 20-MHz and 25-MHz S-25 system specifications were the most important goals for product implementation. To ensure functional correctness, an architecture engineer developed a set of assembly test programs for the floating-point unit. Working from Sparc system specifications and block diagrams in Figures 1 and 2 (on the following pages), the engineer made no assumptions about control implementation during test software development. After completion of the Register-Transfer-Level model of the device, we wrote addi-

0272-1732/90/0200-0055$01.00 © 1990 IEEE



AA	ALU operand A	CMS	Most significant word of the C-port (write port)	Q0	0th entry in instruction queue
AB	ALU operand B	FSR	Floating-point status register	QA0	0th entry in address queue
ALS	Least significant word of the A read port	LS	Least significant	QAN	Nth entry in address queue
AMS	Most significant word of the A read port	MA	MUL oerand A	QN	Nth entry in instruction queue
BLS	Least significant word of the B read port	MB	MUL operand B	RFILE LS	Register file least significant bank
BMS	Most significant word of the B read port	MS	Most significant	RFILE MS	Register file most significant bank
CLS	Least significant word of the C-port (write port)				

Figure 1. Block diagram of the 3170 Floating-Point Coprocessor.

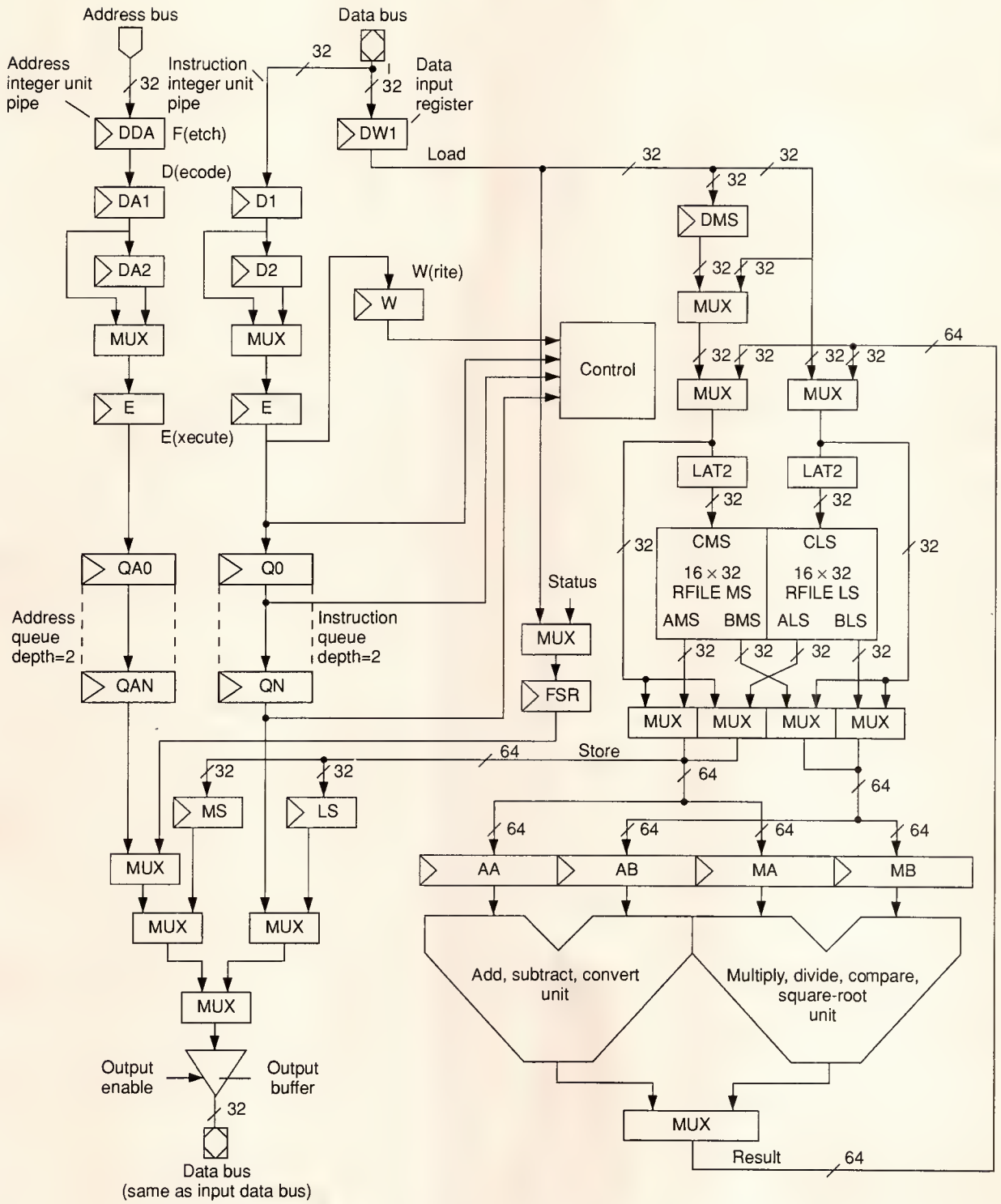


Figure 2. Block diagram of the 3171 Floating-Point Coprocessor.

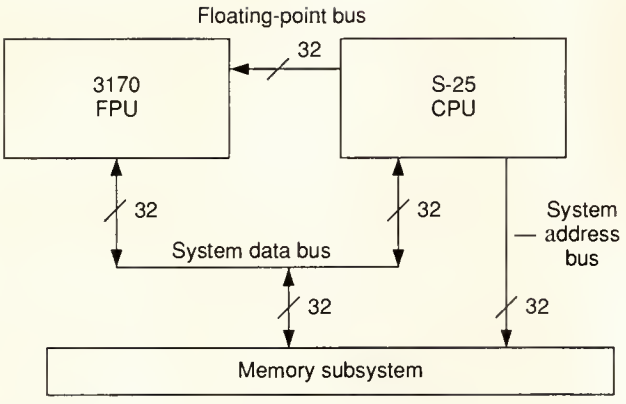


Figure 3. Fujitsu CPU system diagram.

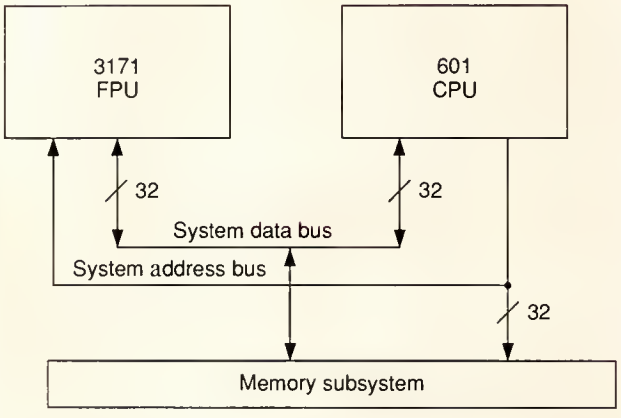


Figure 4. Cypress CPU system diagram.

tional programs primarily to test portions of the coprocessor's design that were most difficult to implement. We achieved better than 95 percent fault coverage before the initial shipments, which indicated the architectural validation's thoroughness. By full qualification, we had attained greater than 98 percent fault coverage.

Since no tool was available to assist in the process, we performed logic partitioning for timing manually. To guarantee that both interfaces met timing specifications, we chose pattern-independent timing simulation of critical paths.

We also required that the S-25 interface function to specification when manufactured on either 1.0- or 1.2-µm processes. This placed additional constraints on the circuit design—many circuit paths had to be fast enough for manufacture on a 1.2-µm process, but not too fast for manufacture on a 1.0-µm fabrication line. For this reason, we designed second-layer metal layout options for switching delay into or out of most critical timing paths.

Having two interfaces in one design intensified our challenge during the initial development stages. Although the 3170 had been our first priority, provisions and hooks to enable efficient 3171 implementation were important from the very beginning. For example, the 3171's very fast output delay and tight setup and hold requirements determined the clocking scheme, pin assignment, and package design for both interfaces.

The selected arithmetic data path determined chip performance characteristics on a cycle-by-cycle basis. Additional performance choices favored design simplicity. For instance, we chose a 64-bit three-port register file because it both simplified control logic and increased performance. Absolutely no complications to the control logic design were acceptable unless they boosted performance or reduced costs significantly.

Chip area was dictated primarily by the 1) selection of the existing arithmetic data path; 2) register file architecture; and 3) gate-matrix automatic layout tools used to help meet our schedule. We used hand layout only for very regular and stable segments of control logic, such as the instruction queue or address comparators for instruction dependency detection. Remaining control logic layout was completely automated, which allowed both logic and timing optimization to continue until the last week of the project.

Bus organization differences

The Sparc architecture specification[4] covers the instruction execution semantics of Sparc-compatible processors. It specifically does not address issues invisible to the programmer. The S-25 and 601 architects each decided to implement the Sparc floating-point instructions in a separate device from the IU, but they chose very different interface mechanisms.

The S-25 IU uses a dedicated bus (the floating-point bus) to transmit instructions and instruction addresses (see Figure 3). Instruction, fetch, decode, and execute details, and the complexities of integer pipeline exceptions and interrupts are concealed from the floating-point unit (FPU). The multiplexed floating-point bus limits the maximum floating-point instruction rate to one per two cycles. However, floating-point instructions take several cycles to execute, and this has a small impact on system performance.

The 601 (see Figure 4) uses a different mechanism under which the FPU must keep a complete copy of the integer execution unit pipeline. As instructions are fetched from memory, the FPU retains the instruction and its address. Control signals generated by the IU allow floating-point instructions to emanate every clock cycle, enabling the FPU to track the instruction pipeline accurately. This nonmultiplexed "copy of the pipeline" technique yields a small increment in system performance, at the cost of a large increase in control logic.

3170/3171 operation

Structures of the 3170 and 3171 are quite similar. Each chip consists of an arithmetic data path, register file, periphery logic around the pads, and control circuitry. Since the 3170 and 3171 implement the same instruction set, the chips share about 75 percent of their design. The arithmetic data path, register file, floating-point queue, and decoding logic are identical for both Sparc floating-point units.

The arithmetic data path consists of two independent modules—the ALU and multiply/divide/square root (MUL/DSQ) units. For the 3170 and 3171, we disabled the two pipeline registers of the ALU and MUL units; pipelining would have complicated the controller without a significant performance improvement. We did, however, keep the ALU and MUL/DSQ units independent of one another and allow them to execute in parallel. Many important algorithms have an equal number of adds and multiplies and are able to take advantage of such parallelism. A parallel ALU and MUL/DSQ architecture is also similar to a previous Sparc implementation with our WTL1164/65 floating-point parts and the controller gate array from Fujitsu. Because of this similarity, the 3170 and 3171 can take full advantage of software previously optimized for the 1164/65 coprocessor design.

The register file comprises two banks of sixteen 32-bit registers, which can be addressed as either 32-bit (single-precision) or 64-bit (double-precision) quantities. The register file has three independent 64-bit ports—two ports to read and one to write. Write and read operations of the same register can occur in one cycle.

In both implementations, a load followed by a floating-point or store operation executes without system hold cycles. For the 3171, this process requires a set of bypass multiplexes on outputs of the register file. If a load happens at the same time that a floating-point operation is finishing, the load takes priority over the instruction in front of the queue. The load instruction updates the register file first, while the floating-point operation completes in the next cycle.

The S-25 and the 601 integer units use a four-stage instruction pipeline to fetch, decode, execute, and write. The hardware pipeline interlocks for more than one-cycle memory accesses in case of an instruction fetch; for multiple-cycle instructions, such as loads and stores; and for data dependencies.

Periphery logic of the 3170 includes a pair of 32-bit registers—one instruction, one address—to support the corresponding IU pipeline for the Write stage. In the 3171, a total of eight 32-bit registers exist—four for instruction and four for address—for the Fetch, Decode, Execute, and Write stages. We also placed a two-entry-deep floating-point queue on the pads. An undecoded instruction and its address are kept on the pads and can be driven out on the data bus during a floating-point store queue operation. Most of the instruction also goes into the control circuit to be decoded. Pipeline registers and floating-point queue entries are duplicated inside the control circuit for the decoded instruction.

From an implementation viewpoint, the number of simultaneous instructions, in addition to radically different timing parameters, is the major difference between the 3170 and 3171. The S-25 coprocessor receives instructions during the Write stage of the instruction pipeline. The 3170 controller has to handle at most one instruction in the instruction pipeline and two floating-point operations in the floating-point queue.

In the 601 implementation, the coprocessor monitors the system address and data buses. It receives instructions, together with the IU, in the Decode stage. The IU initiates floating-point instructions in the Execute stage. The 3171 has to handle up to two operations in the instruction pipeline (one in the Execute stage, and one in the Write stage) in addition to any floating-point operations that may exist in the floating-point queue.

The complexity of handling two simultaneous operations in the instruction pipeline becomes most apparent in the control section, where implementation for the 3171 is about three times more complex than the 3170's random logic. Surprisingly, control timing for the longest critical path in the hardware interlock detection is almost equal for the 25-MHz 3170 and the 40-MHz 3171. About the same amount of activity takes place in one cycle for the 3170 as in one-and-a-half cycles for the 3171. Of course, there are other critical timing parameters unique to the 3171.

On both chips, most control circuitry is dedicated to instruction decoding, floating-point queue of decoded instructions, instruction pipeline state machines, floating-point exception processing, system hold (FHOLD) logic in case of hardware hazards, and arithmetic control state machines. Arithmetic control is completely decoupled from the rest of control circuitry.

The ALU and MUL/DSQ state machines generate corresponding "done" signals upon completion of each instruction. When initiated, each instruction requires a fixed number of cycles that are loaded into the appropriate state machine. Because of the register file's two 64-bit read ports and one 64-bit write port, the overhead for reading operands and writing back the result is only one cycle. Most of the floating-point operation time occurs in the arithmetic units.

Unit operation

We designed both the 3170/3171 chips from a floating-point core unit that could support concurrently two independent ALU operations, two independent MUL operations, and a DSQ operation.

The ALU and MUL units were split into three pipe-

line stages. The first and third stages each took a half cycle to complete. During the first cycle's remaining half cycle, register file read and instruction decoding took place; in the third stage, a register file write and floating-point exception processing occurred. Except for divide and square-root operations, each register-to-register operation took two cycles to complete.

On both the 3170 and 3171, we combined the MUL and DSQ modules into a single MUL/DSQ unit. However, MUL/DSQ and ALU units still act in parallel and can operate simultaneously for independent instructions.

We also disabled pipeline registers for the MUL/DSQ and ALU units. Except for double-precision multiply, single- or double-precision divide, and square root, logic implementation for all arithmetic operations became purely combinatorial. Critical timing paths for the combinatorial arithmetic logic are double-precision add/subtract instructions in the ALU unit and single-precision multiply in the MUL/DSQ unit. Each path takes from 90 to 120 ns, depending on the fabrication process used.

The floating-point functionality of the S-25/3170 and the 601/3171 Sparc implementations with the corresponding operating system software conform completely to the Sparc architecture requirements. These in turn conform to a subset of the *ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic*.[5] Most of the architecture specifications are implemented in the 3170/3171 hardware. The unimplemented aspects of the architecture in hardware included extended-precision operations, handling of NANs (Not A Number), and handling of denormalized numbers on input and/or output of all floating-point operations. The Sparc requirements define four floating-point exception traps—the IEEE, unfinished floating-point instruction, unimplemented floating-point instruction, and sequence error traps. The unfinished and unimplemented traps facilitate correct processing of IEEE-754 standard specifications not implemented in hardware. The 3170/3171 assert an unimplemented trap for the extended-precision floating-point operations and an unfinished trap for all other operations that involve NANs and/or denormalized numbers. If either of the above traps is asserted, the operating system software becomes responsible for the correct instruction execution. The offending instruction is located in the front of the 3170/3171 floating-point queue at the time of any of the floating-point traps.

Multiplier. The multiplier is implemented using a radix-8, modified Booth algorithm.[6,7] It computes a times-three multiple of one of the operands in the beginning of the multiplier operation. Then, a 27×54-bit array of carry-save adders is used twice to form a double-precision multiplication result, and only once for single-precision. A gated clock generated in the multiplier state machine controls iterations through the multiplier array. Final carry propagate addition, IEEE-754 rounding, and normalization occur following computation in the array (see Figure 5). The multiplier generates a special 27-bit result when it operates in a test mode. The 27 bits come from multiplier array bits normally shifted out during the second pass of the double-precision multiplication. This feature facilitates fault-testing of the multiplier with a much smaller pattern set than otherwise would be required. Pattern generation is also much simpler through this test feature.

The multiplier does not treat denormalized numbers and NANs according to the IEEE-754 standard. Operations with the above inputs or multiplies that could produce an abnormal result generate an exception signal near the end of the operation. Such instructions are signaled as unfinished in the Sparc architecture. Floating-point compare instructions are performed in the times-three adder. Sparc move, negate, and absolute value instructions occur in the multiplier as well.

The ALU. The ALU (see Figure 6) performs add and subtract operations as well as format conversion functions between integer, single-precision, and double-precision numbers. Standard steps for a floating-point addition are exponent compare, right shift of an operand with a smaller exponent, mantissa addition, normalization (left shift), and rounding. However, no normalization is necessary for IEEE-754 floating-point numbers if a right shift for more than one bit occurred. At the same time, rounding is unnecessary in the case of normalization by more than one bit. The ALU data path implementation relies on the above IEEE-754 addition properties. Based on the result of an exponent comparison, the ALU critical path either appears as an adder followed by a left shifter, or as a right shifter followed by an adder that also includes rounding logic.

We call the first path "the massive cancellation path" because of the potentially large number of leading zeroes in the result that must be normalized. This path is selected in case of a floating-point subtraction with the operand exponent difference of 0 or ± 1. The second path, chosen if the exponent difference is other than 0 or ± 1, is standard for floating-point addition and subtraction. The resulting implementation is fast and takes little chip area—only one large shifter and a mantissa adder are in the critical timing path for the ALU. At the same time, only one hardware instance of each an adder and a shifter are required for this implementation, as opposed to a two-shifter implementation that is more standard for combinatorial ALUs.

Implementation mapping for the integer- to double-precision floating-point conversion takes place on the "massive cancellation path" in a left-shift-followed-by-no-rounding manner. Floating-point to integer conversion functions map onto the "nonmassive cancellation path"—right-shift-followed-by-rounding.

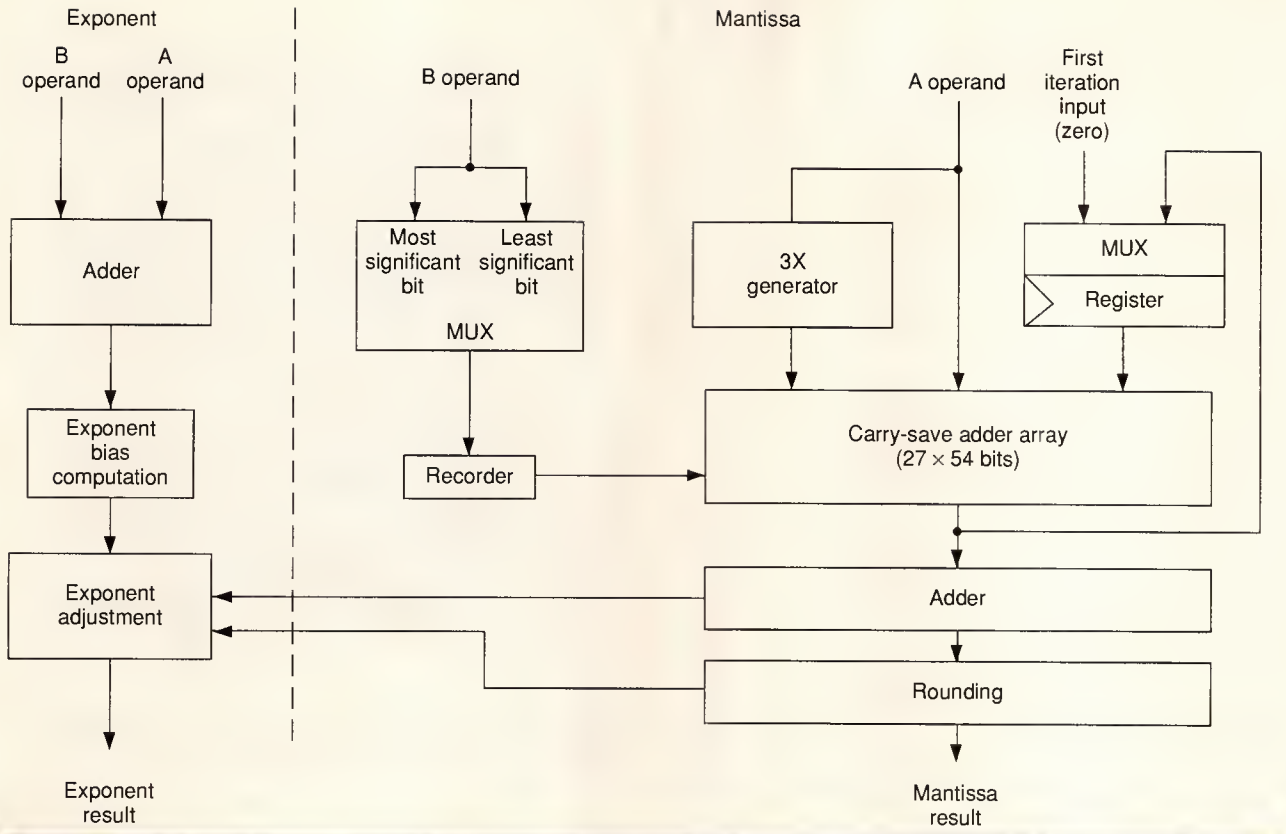


Figure 5. Floating-point multiplier block diagram.

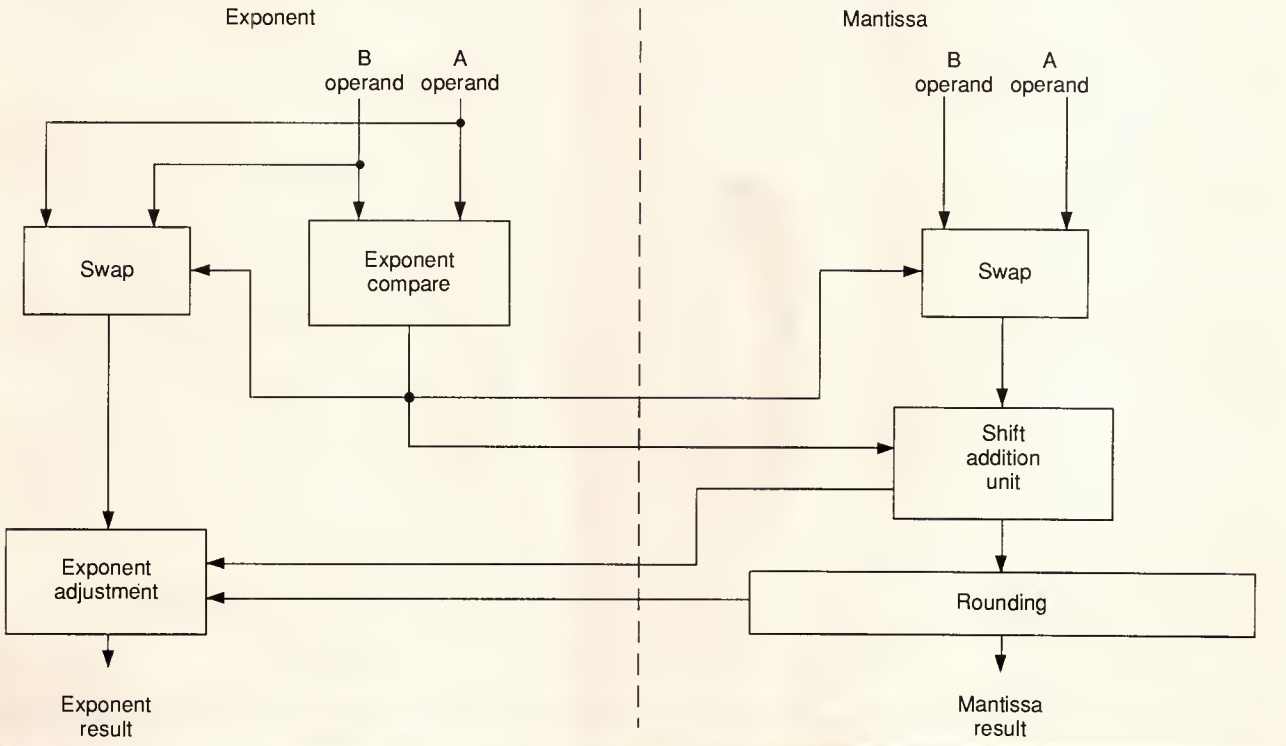


Figure 6. Floating-point adder block diagram.

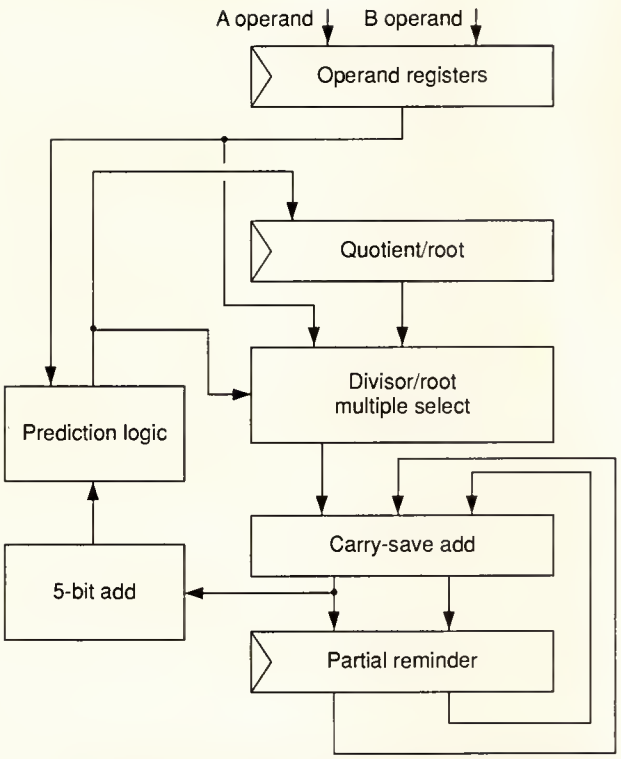


Figure 7. Floating-point divide and square root block diagram.

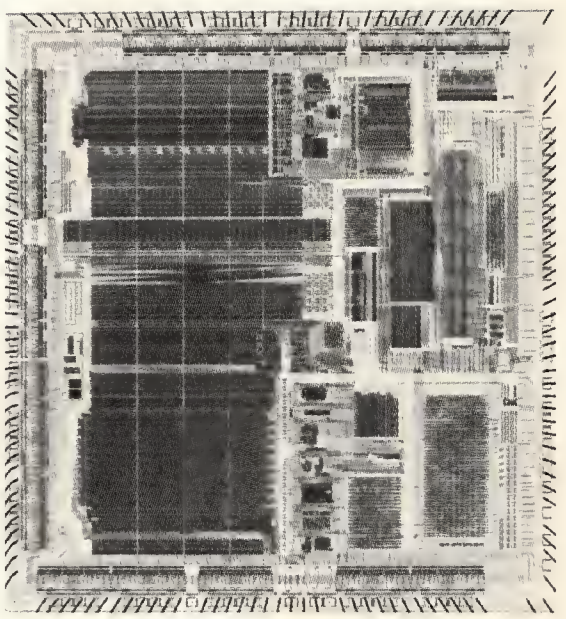


Figure 8. Photomicrograph of the Weitek 3171.

Because it requires a left shift potentially followed by a rounding step, only integer to single-precision conversion does not coexist well with the above architecture. This function executes in two passes through the ALU. The first pass converts an integer to double-precision; the second performs a double- to single-precision conversion. Conversion from single- to double-precision requires only exponent adjustment. Conversion from double- to single-precision requires rounding as well.

The Divide/Square-Root Unit. The original implementation of the divide/square-root unit in the 3 × 64 chip functions independently of the multiplier and the ALU; the MUL and DSQ modules shared only the first division/square-root cycle and final rounding logic. Coupled with the area constraints, this configuration resulted in a selection of iterative algorithms for mantissa divide and square-root computation. The block diagram of the divide unit's mantissa appears in Figure 7. We used the same times-three circuit of the multiplier to implement full redundancy in nonrestoring radix-4 division. Full redundancy in quotient digit representation (±3, ±2, ±1, 0) simplifies digit selection and shortens the unit's critical path. Since it is shared with the multiplier, times-three generation produces no area penalty. The basic divide, square-root iteration consists of a two-level multiplexer for the selection of divisor/root multiples; a one-level carry-save adder; a 5-bit carry-propagate adder for the most-significant bits of the partial remainder; and a two-gate delay prediction logic. Hardware parallelism in prediction logic, full redundancy, and properties of carry propagation allow elimination of lengthy prediction logic delay. The coprocessor clock controls divide/square-root logic. Divide rounding logic is shared with the multiplier.

Choices and results

We implemented both chips in 1.2-µm, double-metal CMOS technology and used a 1.0-µm gate process to produce the 3171's 40-MHz operation. As expected for this Sparc implementation, we found the Fins1/2 to FHOLD and Flush to FHOLD output delays, data bus output delay, and data bus setup and hold time specifications the most difficult to meet for 40-MHz operation.

The 1.0-µm 3171 contains approximately 162,000 transistors packaged in a 143-pin, ceramic grid array. At 33-MHz operation, the 10.1 × 11.4-square-millimeter die produces a 218-mA active current. A photomicrograph appears in Figure 8.

Unlike the accompanying integer units, the 3170 and 3171 do not complete a floating-point computation every clock cycle. Performance is a complex function of clock frequency, number of clocks per operation (operation latency), and the degree to which operations overlap. Table 1 reflects latency of operations.

Provided there is no data dependency, the 3170 and 3171 allow a multiply and an add operation to overlap.

Table 1. Clock cycles per operation in the 3170 and 3171.

Operation	Clock cycles		Nanoseconds	
	3170	3171	3170	3171
Add, subract, convert	5	4	200	100
Single-precision multiply	5	4	200	100
Double-precision multiply	8	6	320	150
Single/double divide	36/64	22/36	1,440/2,560	550/900

Sparc architecture permits no overlapping load and store activity.

One may calculate peak double-precision Linpack benchmark performance by examining the pivot operation's inner loop,[8, 9] which appears inside the Linpack program's DAXPY (double-precision $A \times X + Y$) subroutine. Assuming 100-percent cache hits, the S-25/3170 system requires 63 cycles to execute one iteration of the loop, which contains eight floating-point operations, for a peak performance of 3.2 double-precision Mflops at 25MHz. The 601/3171 system requires only 52 cycles or 6.2 double-precision Mflops at 40MHz. Naturally, performance will degrade when wait states are inserted due to cache misses. Since approximately 80 percent of total CPU time for the benchmark occurs in this subroutine, it is a good indicator of the overall benchmark performance.

The 3170/3171 project progressed from start to shipment in nine months. Central to this timetable was the elimination of risk at every step. In addition, we combined functional simulation with an extensive test suite created independently of implementation. Static timing path simulation guaranteed quick, reliable parametric verification. Adopting the floating-point data path, largely intact, from an existing product, we used automated placing and routing tools extensively. We followed proven production processes and, above all, we maintained a keep-it-simple, get-it-right-the-first-time attitude. As a result the first mask set was qualified, unchanged, for full volume production.

References

1. Fujitsu Limited and Fujitsu Microelectronics, Inc., "Sparc MB86901 (S-25) High Performance 32-bit RISC Processor," Product Description, San Jose, Calif., Sept. 1988.
2. Cypress Semiconductor and Sun Microsystems, Inc., *CY7C600 RISC Family Users Guide*, San Jose, Calif., June 1988.
3. Weitek Corp.,"WTL3164/WTL3364 Floating-Point Datapath" data sheet, Sunnyvale, Calif., April 1988.
4. Sun Microsystems, Inc., "The Sparc Architecture Manual Review Draft Version 7 Plus Proposed Floating-Point Changes, Revision B," Mountain View, Calif., July 5, 1988.
5. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*, IEEE Computer Society Press, Los Alamitos, Calif., 1985.
6. A. D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanics and Applied Mathematics*, Vol. 4, Part 2, 1951.
7. S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, CBS College Publishing, New York, 1982.
8. Weitek Corporation, "WTL3170 Sparc Floating-Point Coprocessor," data sheet, April 1989.
9. Weitek Corporation, "WTL3171 Sparc Floating-Point Coprocessor," data sheet, July 1989.

Mark Birman is an IC design manager at Weitek who has been associated with several floating-point building block and coprocessor designs. As a student engineer, he worked at the IBM Yorktown Research Center and at the Stanford Linear Accelerator Center.

Birman received a BSEE and an MSEE from Stanford University. He is a member of Phi Beta Kappa and Tau Beta Pi.

Allen Samuels, manager of the Architecture Group, has worked in software and hardware development in the mainframe, minicomputer, and microcomputer industries. He holds responsibility for development of floating-point coprocessors, graphics processors, and RISC embedded controllers. He received a BSEE from Rice University and is a member of the IEEE and ACM.

WTL 3170/71

George Kai-Gene Chu, a senior design engineer, has contributed to many key floating-point coprocessor designs and patents. He received his BS degree in Shanghai, China, and holds an MSEE degree from North Carolina State University.

Ting Chuk is a design engineer, and is responsible for high-speed floating-point design and for design and development of Sparc coprocessors. Her previous position was as a VLSI design engineer with Xerox Corporation. She received her BSEE degree in computer science from the University of California at Berkeley.

Larry Hu is a design engineer. His interests include floating-point processor and coprocessor design and development. He holds a BSEE degree from Stanford University, where he studied computer architecture and logic design.

John McLeod is a member of the Architecture Group, where he is responsible for new products' specification, development, and verification. McLeod received a BSc in engineering from Lanchester Polytechnic, Rugby, England, and an MSc in bioengineering from Strathclyde University, Glasgow, Scotland.

John Barnes is vice president of IC Development. He is responsible for the architecture, logic and circuit design, CAD, and system engineering groups. He has held previous positions in circuit design and process development at Motorola, Mostek, Fairchild, and AMI.

Barnes holds an MSEE degree from the University of California at Berkeley and a PhD from the University of Michigan, Ann Arbor. A senior member of the IEEE, Barnes has served several times on the society's IEDM, ISSCC, CICC, and VLSI Technology program committees.

Direct any questions concerning this article to Mark Birman at Weitek Corporation, 1060 East Arques Avenue, Sunnyvale, CA 94086.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 150 **Medium** 151 **High** 152

READER-SERVICE CARD

IEEE MICRO

February 1990 issue (card void after August 1990)

Name _____

Title _____

Company _____

Address _____

City _____ State_____ ZIP _____

Country _____ Phone (_____) _____

Please send (Circle those you want)

201 Publications catalog
202 Membership information
203 Student membership information
204 Senior membership information
205 *IEEE Micro* subscription information

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150 151 152 177 178 179
153 154 155 180 181 182
156 157 158 183 184 185

159 160 161 186 187 188
162 163 164 189 190 191
165 166 167 206 207 208

168 169 170 209 210 211
171 172 173 212 213 214
174 175 176 215 216 217

Product Information 1
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	—
11	31	51	71	91	111	131	—
12	32	52	72	92	112	132	192
13	33	53	73	93	113	133	193
14	34	54	74	94	114	134	194
15	35	55	75	95	115	135	195
16	36	56	76	96	116	136	196
17	37	57	77	97	117	137	197
18	38	58	78	98	118	138	198
19	39	59	79	99	119	139	199
20	40	60	80	100	120	140	200

READER-SERVICE CARD

IEEE MICRO

February 1990 issue (card void after August 1990)

Name _____

Title _____

Company _____

Address _____

City _____ State_____ ZIP _____

Country _____ Phone (_____) _____

Please send (Circle those you want)

201 Publications catalog
202 Membership information
203 Student membership information
204 Senior membership information
205 *IEEE Micro* subscription information

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150 151 152 177 178 179
153 154 155 180 181 182
156 157 158 183 184 185

159 160 161 186 187 188
162 163 164 189 190 191
165 166 167 206 207 208

168 169 170 209 210 211
171 172 173 212 213 214
174 175 176 215 216 217

Product Information 2
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	—
11	31	51	71	91	111	131	—
12	32	52	72	92	112	132	192
13	33	53	73	93	113	133	193
14	34	54	74	94	114	134	194
15	35	55	75	95	115	135	195
16	36	56	76	96	116	136	196
17	37	57	77	97	117	137	197
18	38	58	78	98	118	138	198
19	39	59	79	99	119	139	199
20	40	60	80	100	120	140	200

SUBSCRIPTION CARD

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑**YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only **$19** for a year's subscription (six issues).

Society: _____ IEEE membership no.: _____

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate $9.50 for three bimonthly issues.

❑**YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE (but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other qualified professional technical society, pay the sister-society rate of only **$35** for a year's subscription (six issues).

Organization: _____ Membership no.: _____

Full Signature *Date*

Name

Street

City

State/Country *ZIP/Postal Code*

❑ Payment enclosed

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

Expiration date

Month *Year*

Prices valid through 12/31/90
2/90 Micro

Change orders also taken by phone:
(714) 821-8380 8a.m. to 5 p.m. Pacific time
Circulation Dept.
PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: _____

_____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

Reviewers needed. *If interested, send professional data to Joe Hootman, EE Dept., University of North Dakota, PO Box 7165, Grand Forks, ND 58202.*

PO Box is for reader-service cards only.

PLACE POSTAGE HERE

IEEE Micro
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

For reader-service inquiries, see other side.

Reviewers needed. *If interested, send professional data to Joe Hootman, EE Dept., University of North Dakota, PO Box 7165, Grand Forks, ND 58202.*

PO Box is for reader-service cards only.

PLACE POSTAGE HERE

IEEE Micro
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
Circulation Dept.
PO Box 3014
Los Alamitos, CA 90720-9804
USA

ISMM

CALL FOR PAPERS

ISMM International Conference

INDUSTRIAL, VEHICULAR AND SPACE APPLICATIONS OF MICROCOMPUTERS

October 10-12, 1990
New York, U.S.A.

SPONSOR

The International Society for Mini and Microcomputers - ISMM.
Techical Committee on Computers.

LOCATION

New York Penta Hotel

SCOPE

- Control
- Communication
- Power systems
- Air traffic controller
- Signaling
- Instrumentation
- Signal processing
- Remote sensing
- Image processing
- Manufacturing
- Pattern recognition
- Data acquisition
- Data processing
- Simulation
- Navigation
- Guidance
- Robotics
- Expert systems
- Monitoring
- Protection
- Surveying
- Others

SUBMISSION OF PAPERS

Three copies of 1000 word short version of the paper to be received by **May 1, 1990**. Please give a maximum of five key words describing the contents of your paper. Notification will be mailed by **June 1, 1990**. The full papers in camera-ready form together with preregistration are due **September 3, 1990**.

Authors wishing to have their papers considered for possible publication in the ISMM Journal Microcomputer Applications should send three copies of their expanded papers directly to Professor L. Miller, Department of Computer Science, Iowa State University, Ames, Iowa 50011 U.S.A. The Editor also welcomes papers not submitted for possible presentation at the ISMM conference.

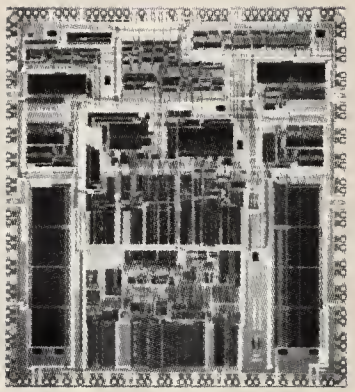
INTERNATIONAL PROGRAM COMMITTEE

O.A.L. Badr	Saudi Arabia	J.Gil	Mexico	E.Luque	Spain
M.S.Fadali	U.S.A.	J.L.Houle	Canada	G.Mastronardi	Italy
B.Furht	U.S.A.	G.K.F. Lee	U.S.A.	L.Miller	U.S.A.
L.Furin	U.S.A.	Y.Luqing	P.R.C.		

ADDRESS

For submission of papers and to be placed on the mailing list, write to: MICRO'90 NY, ISMM Secretariat, P.O.Box 25, Stn. G., Calgary, Alberta, Canada. Tel.(403)270-3616. Fax.(403)270-8855.

FEATURE



The 68040 Processor

Part 1, Design and Implementation

In the first of a two-part series, the design team explains its total approach and the workings of the integer and floating-point units.

Robin W. Edenfield
Michael G. Gallup
William B. Ledbetter, Jr.
Ralph C. McGarity
Eric E. Quintana
Russell A. Reininger

Motorola

The 68040 is a third-generation, full-32-bit microprocessor in the Motorola 68000 family. The 68040 integrates over 1.2 million transistors on one chip and can execute the complete 68020 microprocessor and 68882 floating-point coprocessor instruction sets. Its sustained performance level is four times that of the 68020 and 10 times that of the 68882 (all with the same clock frequency of 25 megahertz). Pipelined integer and floating-point execution units that operate concurrently with separate internal memory controllers and an autonomous bus controller contribute to this performance level. Physical caches of 4 Kbytes each for instruction and data reside on chip. Separate address-translation caches of 64 entries apiece operate in parallel with the instruction and data caches. This arrangement provides complete memory management in a virtual, demand-paged operating system. Here we present the design of this microprocessor and the trade-offs that we made in the process. (Figure 1 summarizes the processor.)

The major design goals in order of priority were

- compatibility,
- performance,
- schedule, and
- integration.

Because the 68040 joins a family of installed microprocessors, these fundamental goals primarily resulted from user input—and our perceived evolution of all microprocessors—over the term of the 68040 design.

We defined *compatibility* as the capability to execute all user-mode code without change. However, we realized that some changes to the supervisor code were inevitable due to the characteristics of the processor. Large physical cache, memory management unit (MMU) control, and chip integration all would require that the operating system "recognize" the difference between previous-generation processors and the 68040. This factor led us to allow deviations in the supervisor model that improved either performance or on-chip silicon requirements.

Market conditions required the processor to deliver four times the performance of the 68020 at the same clock speed, with working silicon implemented in 1989.

0272-1732/90/0200-0066$01.00 © 1990 IEEE

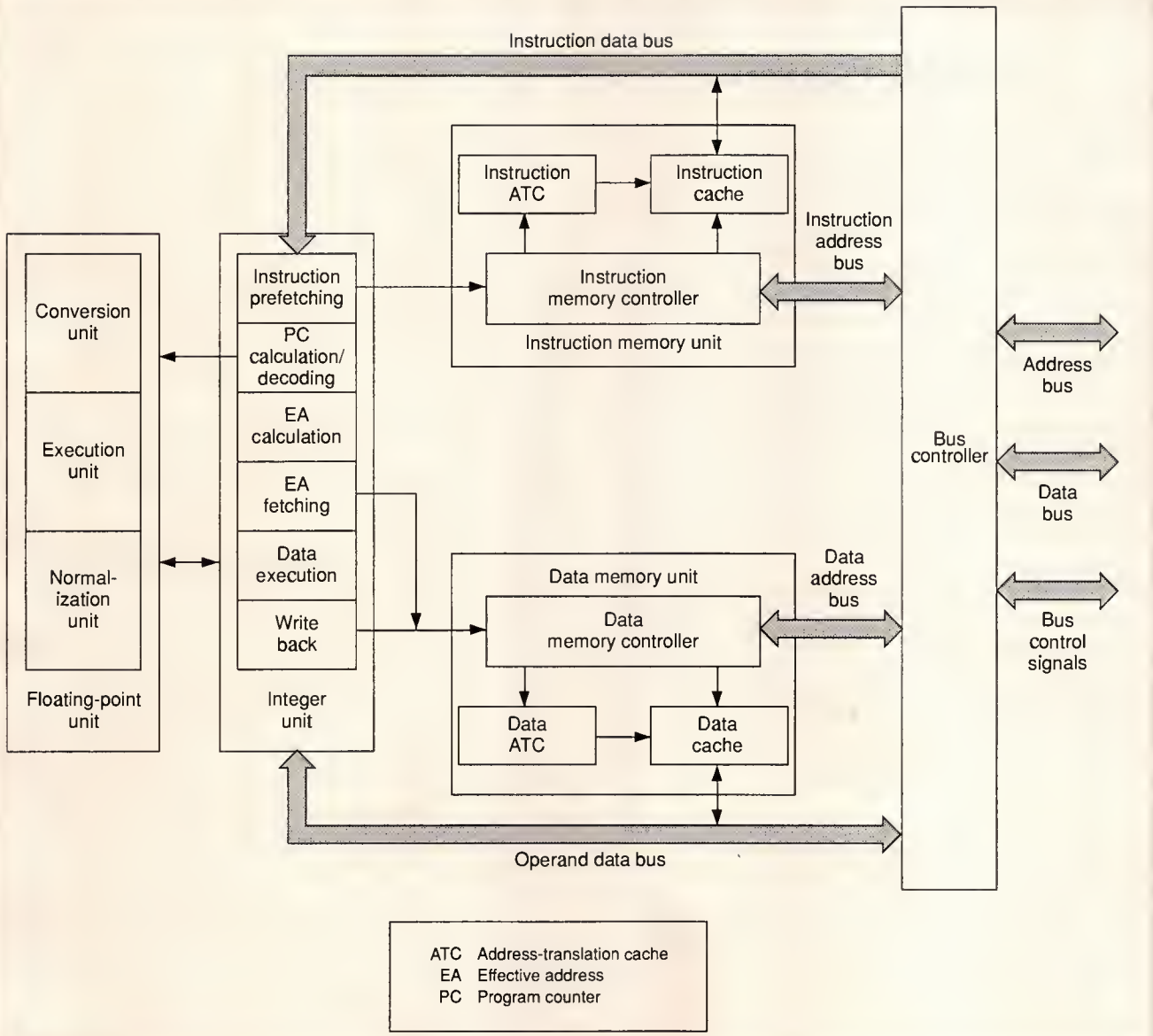


Figure 1. Block diagram of the 68040.

Our integration goal was to achieve the combined effect of separate 68000 family components: integer processing, on-chip floating point, and on-chip memory management capable of supporting a demand-paged operating system.

We followed a different process for this design from that traditionally performed at Motorola. The major changes were a heavy emphasis on early user input before and during the specification process and the use of functional simulation instead of breadboarding to verify the design. From the design goals, we presented an initial design specification to users. We worked with the users to refine the requirements and the architecture to produce the final specification.

Concurrently, other members of the design team studied the 68020 and 68882 instruction sets as used in system designs. Together we built a system to collect bus activity from a 68020 processor. With this system, we accumulated traces of over 60 million bus cycles of real application code in execution.

To aid in the analysis of the bus-cycle trace data, we created a high-level, statistical performance model. This model used parameters obtained from the traced bus cycles and disassembled instruction statistics. The model—along with a cache and an MMU simulator driven from the trace data—allowed us to quickly determine the effects of such design decisions as

Cache Modes

Data caches generally operate in one of two modes: write through or copy back.[1] On a cache hit, a write-though cache updates the data in the cache and writes the data to main memory via the external bus. The system keeps main memory up to date, or coherent, with the data in the cache. Because a write-through cache generally does not allocate on a write miss, the only way for new data to be placed in the cache is on a read miss.

For a write hit to a copy-back cache, the system updates the data in the cache, but does not update main memory. This situation creates "dirty" data in the cache. The only valid location for the data is in the cache since the memory version of the data is incorrect. Main memory is called incoherent when there is dirty data in the cache. Note that any read hit accesses the dirty data from the cache, not main memory, so that data integrity is preserved. The system only writes (pushes) a dirty cache entry to main memory when it needs to remove the entry to make room for another one. Copy-back caches generally allocate on a write miss.

- physical cache size and associativity,
- address-translation cache size and associativity,
- MMU table-search algorithm selection,
- branch methods and branch-target buffer inclusion,
- MMU page size,
- cache write methods, and
- bus clock speed.

Philosophy

The first major decisions we made were to optimize those instructions in the 68020 that executed most frequently and to allow other instructions to be implemented conveniently. We picked a reasonable subset of the 68020 instruction set based on trace data and user input. We examined those instructions and discovered that most of them would execute in one clock cycle if the IU were pipelined. (A clock cycle here refers to the input clock to the chip—which is also used by the system—running at 25 MHz.)

Along with this analysis, we decided that for performance reasons the overall cache architecture should continue to be based on the Harvard model (separate address and data spaces), as in the 68030. Further, we matched the integer-ALU cycle time to the cache-access time. This procedure allowed optimized instructions to access each cache once per clock cycle and fixed the peak instruction-execution rate to the ALU cycle rate. The entire internal and external memory design rested on these decisions.

External memory systems that feature either low cost or high performance were important to users. Consequently, the 68040 bus design maximized the performance available from low-cost memory systems (dynamic RAMs) while minimizing the degradation to high-performance designs (the fastest static RAMs). Therefore, we interfaced the bus to standard transistor-transistor logic (TTL) devices and simplified the protocol to minimize the latency from cache misses of required data.

Users predicted that most of the designs in the 68040-generation time frame would only use one processor for mainline computing. Consequently, we optimized both the internal cache operations and the external bus protocol for the uniprocessor case. We determined the data-cache write method based on principles of performance in uniprocessors versus requirements for shared memory access in multiprocessors. The uniprocessor case requires a copy-back write policy with read and write allocation for maximum performance and minimum bus utilization (see the accompanying box).

However, the multiprocessor case requires complicated bus and cache-state protocols with copy-back data. Also, several key users differed in the approach they wanted in a multiprocessor coherency scheme. These complications led us to add a write-through mode to the cache to specifically support multiprocessing. The write mode of the data cache—copy back or write through—can be set on a page basis by a field within the page descriptor used by the MMU. In write-through mode, the cache controller only allocates on a cache-read miss. Writes proceed directly to the bus and do not allocate a new entry in the cache. In copy-back mode, the cache allocates an entry in the cache on a miss and requests a line read on the bus, regardless of whether the integer unit requested a read or a write to memory.

Integer unit

The 68040 integer unit, or IU, executes the most common instructions in one clock cycle while maintaining user-code compatibility with the 68000 family. As discussed, only supervisor-mode instructions controlling the caches and MMUs differ from previous implementations.

To increase performance in the IU over previous-generation parts, the 68040 reduces both the number of ALU clock cycles per instruction and the ALU cycle time. For a 25-MHz 68020, the ALU cycles at 12.5 MHz, whereas the 25-MHz 68040 ALU cycles at 25 MHz.

Table 1.
A comparison of common effective-address instructions in the 68020 and 68040.

Instruction	Addressing mode	25-MHz clock cycles 68020	68040	Operation
Move	Rn,Rn	2	1	Register-to-register move
Move	<OEA>,Rn	6	1	Memory-to-register move
Move	Rn,<OEA>	6	1	Register-to-memory move
Move	<OEA>,<OEA>	8	2	Memory-to-memory move
Move	(An,Rn,d8),Rn	10	3	Memory-indexed source
Move	Rn,(An,Rn,d8)	6	3	Memory-indexed destination
Move Multiple	RL,<OEA>	$4 + 2n$	$2 + n$	Move multiple registers to memory
Move Multiple	<OEA>,RL	$8 + 4n$	$2 + n$	Move memory to multiple registers
Simple Arithmetic	Rn,Rn	2	1	Register-to-register arithmetic
Simple Arithmetic	Rn,<OEA>	6	1	Register-to-memory arithmetic
Simple Arithmetic	<OEA>,Rn	8	1	Memory-to-register arithmetic
Shifts	—	4	2	Shift 1 to 31 bits
Branch Taken	—	6	2	—
Branch Not Taken	—	4	3	—
Branch to Subroutine	—	6	2	—
Return from Subroutine	—	10	5	—

To reduce the number of clock cycles per instruction, we examined the previously mentioned trace data. From this data, we derived dynamic instruction frequency. Analysis of the instruction frequency led to the selection of a pipelined architecture to reduce load and store operations to one clock cycle. Pipelining allowed the 68040 to optimize the most common effective addressing modes:

- register direct, or Rn;
- address-register indirect, or (An);
- address-register indirect with postincrement, or (An)+;
- address-register indirect with predecrement, or –(An);
- address-register indirect with 16-bit offset, or (An,d16);
- absolute, or $Address; and
- immediate, or #Data.

These optimized effective-address (<OEA>) calculations and address fetches execute with no penalty for instructions such as logic, arithmetic, and data movement. Table 1 compares cycle times for the most common instructions with these effective addresses in the 68020 and 68040. (RL stands for multiple register list, and d8 refers to an 8-bit offset).

The more complex instructions that are not in the optimized set generally execute in less than half the number of 25-MHz clock cycles in a 68040 than they do in a 68020 or 68030.

Integer pipeline

The 68040 has a six-stage integer pipeline (Figure 1). Figure 2 on the next page details these stages:

- instruction-prefetching (IP),
- program-counter calculation and decoding (PCD),
- effective-address calculation (EAC),
- effective-address fetching (EAF) for operands,
- data-execution (DE), and
- write back (WB).

The implementation has three independent controllers: the PC controller for the IP and PCD stages; the effective-address (EA) controller for the EAC, EAF, and WB stages; and the DE controller for the DE stage. The PC controller is a finite state machine that relies on instruction-length and change-of-flow (branch, jump) decoding controls in the IP and PCD stages. The EA controller combines PLA decoding and microcode sequencing.

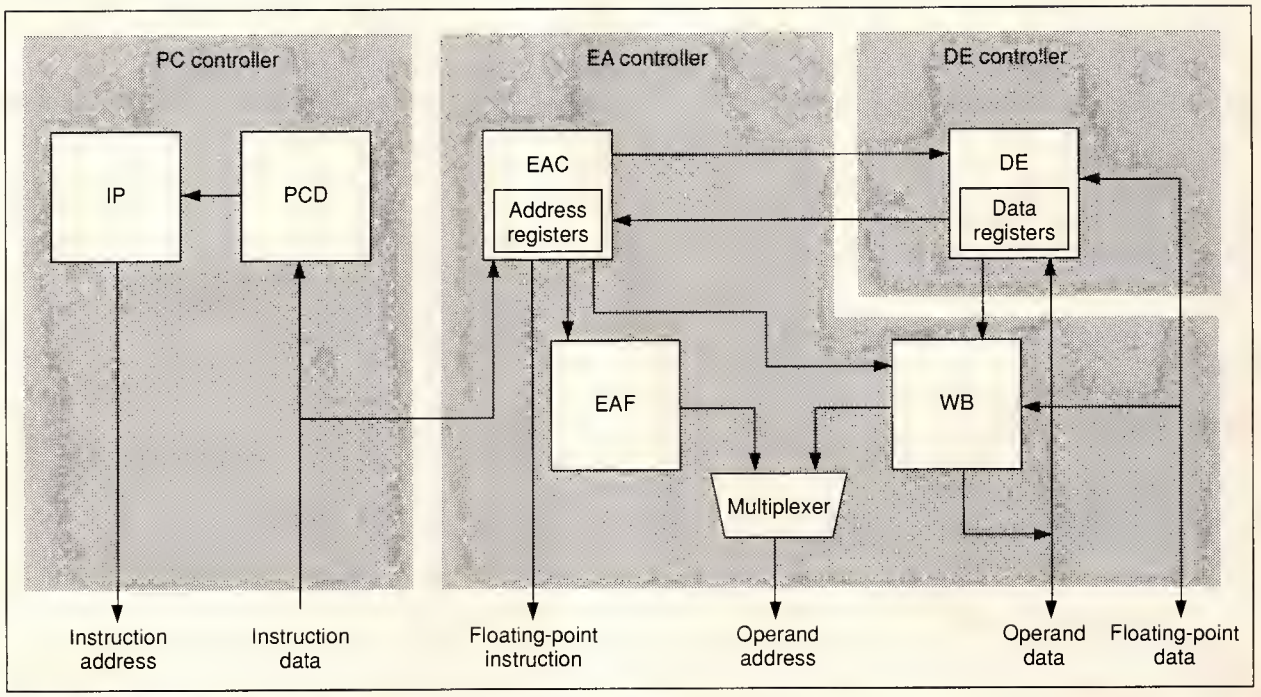


Figure 2. Integer pipeline of the 68040.

The first cycle of execution in the EAC stage receives control from a PLA decoder (not shown). A microcoded sequencer contained in the EAC stage subsequently controls this stage, and a second microcoded sequencer controls DE clock cycles. For most instructions, the EAC and DE stages work independently to allow maximum instruction-execution overlap. Extra cycles spent in each stage due to complex functions or cache misses can be fully overlapped by another stage. For complex instructions such as Move Multiple (MOVEM), the EAC and DE stages work together. The EAC stage generates a new load or store address each cycle, and the DE stage handles data movement to or from the register files.

The IP stage maintains a full 8-word (128-bit) instruction buffer to support execution of one-cycle, 3-word instructions indefinitely. As required, half of the instruction buffer is loaded with 64 bits from the instruction cache. The system can access the instruction cache every cycle to load the buffer.

The PCD stage selects 48 bits (3 sequential instruction words) from the buffer each cycle for instruction decoding and immediate data extraction. The 16-bit opcode and 6 bits from the first extension word provide for integer and floating-point instruction decoding. The 2 extension words provide for immediate and branch-displacement values. The PCD stage has separate instruction-length decoding to allow the PC to be incremented by 1, 2, or 3 words and to select a new 3-word instruction from the instruction buffer every clock cycle. The IU pipelines decoding for the EAC and DE stages and makes it available in the appropriate cycle.

The EAC stage calculates load-and-store effective addresses. It contains the address registers and controls them completely. This fact allows address-register loading and addition to complete early if the source operands are immediate values or other address registers. Early completion reduces stalls from address-register blockage for subsequent instructions that use the modified address register in an address calculation. The DE stage can update address registers if a source operand is either in memory (cache or main) or in a data register. Address-register updates by the DE stage result in only one clock cycle of address-register blockage if the next instruction uses the modified address register. Complex instruction addressing modes, such as memory indirection, require more than one clock cycle to complete. The EAC microcoded sequencer controls the second and succeeding cycles of these addressing modes.

The DE stage performs data-register operations and transfers data to the memory write-back buffer. The DE stage controls the data registers, condition codes, ALU, and barrel shifter (not shown). The DE also handles communication with other major functional units on the 68040 during exceptional conditions and memory-management table searches.

The EAF and WB stages provide for memory load-and-store accesses. These stages are controlled by combinational logic. When the EAC stage writes the

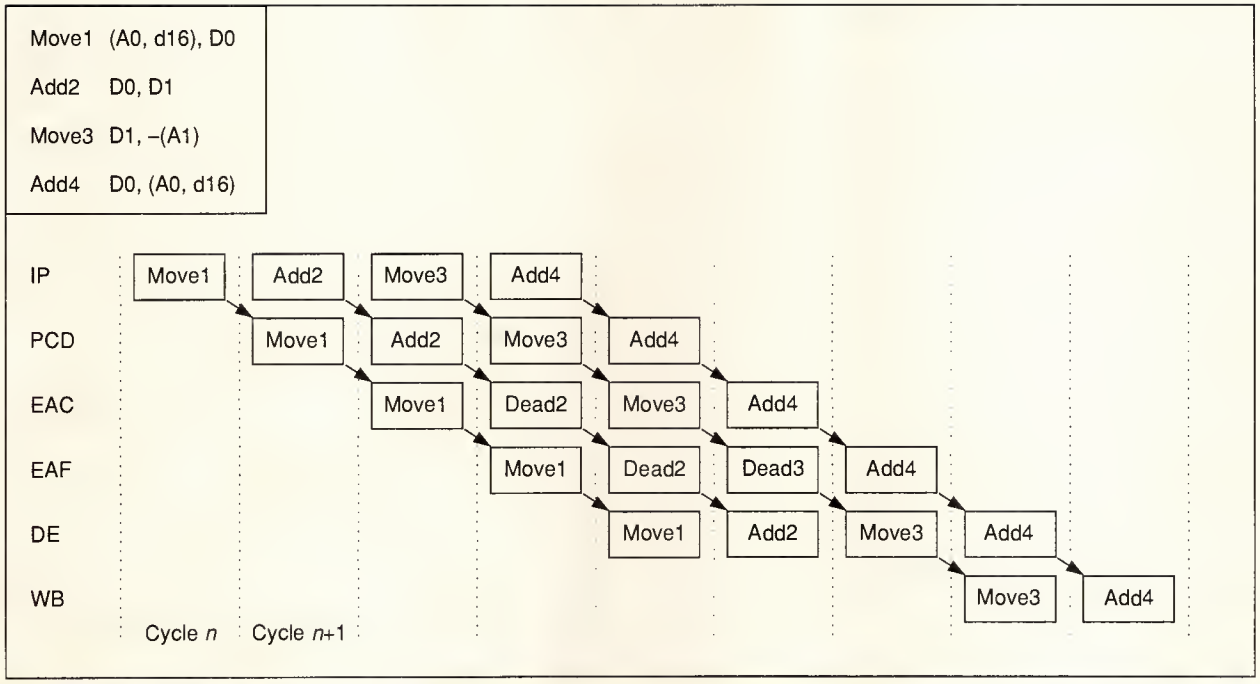


Figure 3. Integer instruction flow for typical optimized instructions in the 68040.

EAF address buffer, the EAF automatically initiates a load. A write-back address queue with an address register for each of the remaining stages of the pipeline facilitates store operations. For instructions that write to memory, the EAC loads the first register in this queue. When the DE stage completes execution of an instruction that requires a store operation, it loads the write-back data buffer. This action causes the WB stage to initiate a store to the cache and the queue to advance.

Since only one load or store operation can be initiated in each cycle, the EAF and WB stages must arbitrate for the data cache. Loads from the EAF stage normally have priority over stores from the WB stage except when the logical address for the load and store match. This match prevents a read-before-write hazard. The WB stage also gains priority if the DE requires a data load into the WB data buffer for a second store operation. A store can be buffered in the final WB stage indefinitely without impacting execution of subsequent instructions.

An underlying reason the 68040 architecture has dedicated EAC and EAF states is that 40 percent of all instructions have data-read accesses. With pipelining, most read accesses have no penalty. Dyadic instructions (Add, SUB, And, Or, EOR) that have a memory operand as either their source or destination execute in one clock cycle. Also, address register updates occurring in the EAC stage eliminate pipeline blockage for instructions with postincrements, predecrements, or immediate adds/loads to address registers.

Figure 3 demonstrates the pipeline operation by showing a typical sequence of instructions. Each column in the figure represents one 25-MHz clock cycle. The notation "dead" means that no useful activity occurs at that pipeline stage.

Branches

The 68040 improved branch performance relative to the 68020 and 68030 from six taken/four not-taken clock cycles to two taken/three not-taken cycles (ignoring cache-miss and cache-line-alignment effects). This performance occurs despite the deeper pipeline in the 68040. We optimized branches on the 68040 for the branch-taken case since trace data shows that 75 percent of all branch instructions (that is, Conditional Branch, Unconditional Branch, Branch to Subroutine, and Decrement and Branch) are taken. A dedicated branch adder in the PCD stage adds the branch displacement to the PC in every cycle. This addition is speculative as it occurs before the branch has been decoded. A dedicated branch-instruction decoder controls the processes of loading the result of the branch adder into the PC and starting a change-of-flow sequence in the PCD stage. If the instruction is not a branch, the IU ignores the result of the addition operation.

When a conditional change-of-flow sequence is started (a Conditional Branch or a Decrement Branch), the not-taken PC, the not-taken EAC decoded instruc-

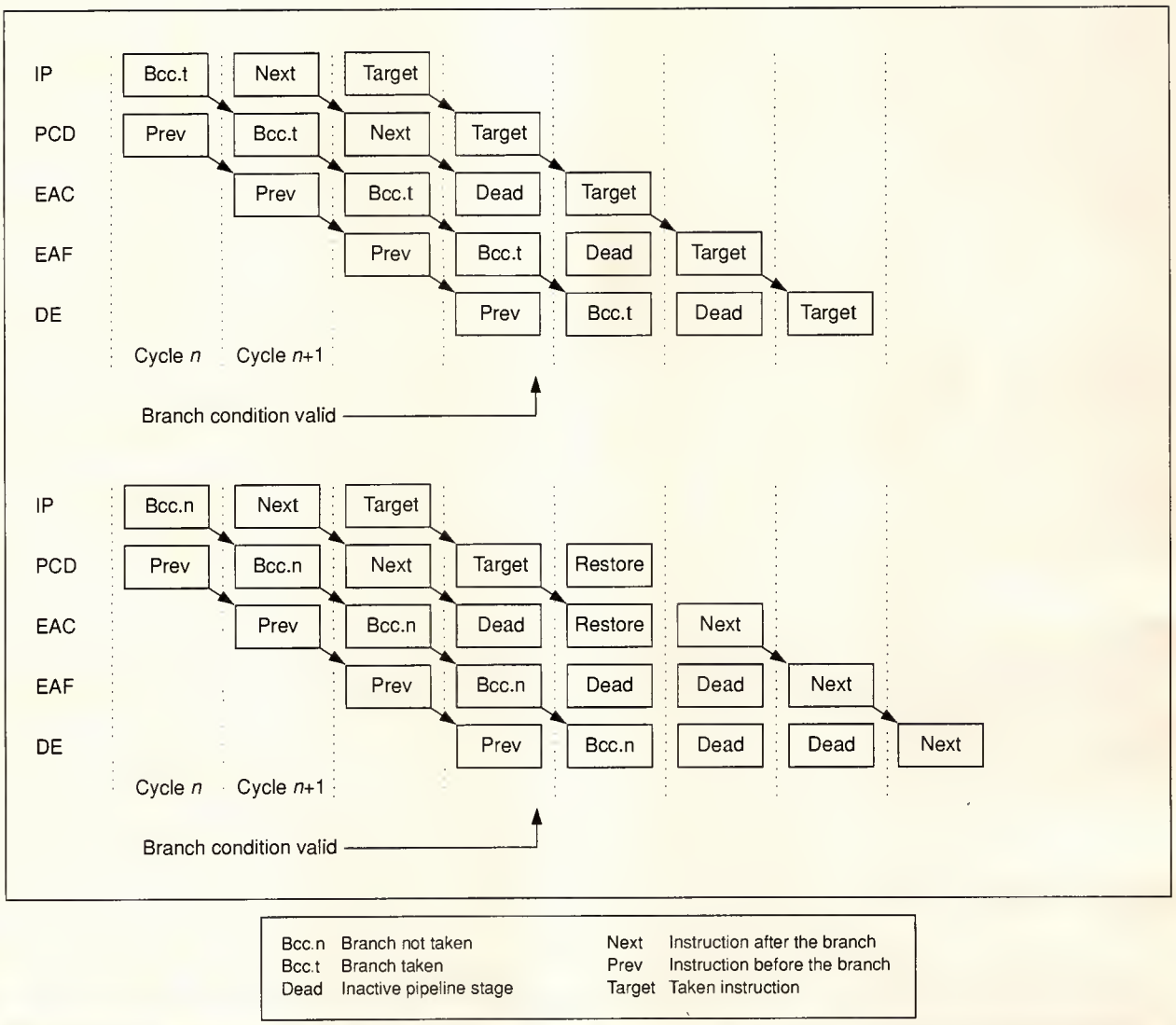


Figure 4. Integer instruction flow for taken (top) and not-taken (bottom) branch instructions.

tion, and the instruction buffer are saved in special shadow registers. As shown in Figure 4, the condition codes from the instruction preceding the branch are valid at the same time the first opcode stream has been decoded. This timing allows execution to start in the EAC stage for the target in the next clock cycle. If the condition is not taken, one cycle is required to recover the not-taken path from the shadow buffers.

We evaluated some design trade-offs for branch-instruction time minimization: 1) changing the architecture by adding delayed branches, 2) optimizing not-taken branches, and 3) adding a branch target buffer (BTB).[2] We rejected delayed branches because changes to the architecture would not help existing code. Optimizing not-taken branches created a problem of partially executed instructions in the EAC and EAF stages that had to be undone, which could result in clock-cycle penalties to the taken path. We rejected the inclusion of a 32-entry BTB based on issues of die size and circuit complexity. A one-cycle improvement in correctly predicted branches improved system performance by only 5.7 percent, which was not enough to justify the silicon-area investment. Also, system and circuit designs of the components to load, maintain, and clear the BTB were so complex they would have slowed the schedule.

Another interesting trade-off possibility existed. Should the PCD and EAC operations occur in one merged stage or in two separate ones? If the EAC and PCD states were united, fewer pipeline stages would separate them. This circumstance would allow an earlier resumption of prefetching for instructions that follow the instruction after the branch, lessening the chance of prefetching-induced stalls. However, merging the EAC and PCD stages would mean that far fewer

instructions would possess optimized (one-clock-cycle) effective-address calculations, since EAC-optimized decoding would only have part of one clock cycle to complete.

Since every prefetch operation obtains an entire half line of instructions, the IU prefetches enough of the instruction stream prior to the execution of the branch so that the extra stage imposes no penalty. We created a statistical model based on the average length of optimized instructions to determine whether the extra stage causes a stall. From it we found that the extra pipeline stage adds only 0.36 of a clock cycle to not-taken branches (making them 3.36 clock cycles). The increased number of optimized instructions provided by the extra pipeline stage more than compensates for this partial increase in clock cycles.

Integer exceptions

Exception processing in the 68040 differs from that of the 68020 and 68030 only in terms of memory-access faults. Traps, interrupts, and tracing are identical to the 68020 and 68030 programmer's model. For memory-access faults (bus-error and page), the existence of a restart exception model marks a change from the continuation exception model used in the 68010, 68020, and 68030. We selected the restart model because it needs fewer bytes to save exception information than a continuation model does (62 versus 250). A continuation model would have resulted in much slower exception stacking and return execution, as well as increased control and routing to access the state information on chip. However, the restart model requires the capability to "back out" of partially executed instructions.

When a memory-access error occurs (page fault or physical bus error), the 68040 stops execution of the current instruction and writes to the exception stack. This write contains the necessary state information for the operating system to correct the fault and return control to the current instruction stream. If an instruction has partially executed when the exception occurs, the state of the processor prior to starting the instruction is restored. The exception appears to the programmer's model to have occurred on an instruction boundary. The access-error stack frame contains

- the return instruction address,
- the faulting access address,
- the faulting access bus attributes,
- special case flags, and
- address and data for up to three pending memory writes.

The operating system completes pending writes.

The 68040 exception model has some special continuation cases. The Move Multiple instruction can corrupt address registers, data registers, and memory locations used in calculating the effective address. Restoring the state of the processor and memory prior to starting the instruction was not feasible, and simply restarting a MOVEM instruction could result in calculating a different effective address result. To solve this problem, an access error during a partially executed MOVEM stacks the calculated effective address and sets a flag on the stack frame. The Return from Exception instruction (RTE) reads the flag and the effective address on the stack frame. When the MOVEM instruction restarts, the IU skips the effective-address calculation and reads the effective address from the stack frame.

The other cases of continuation concern memory-access errors that are handled before instruction-generated exceptions. The special cases are trace exceptions, unimplemented floating-point exceptions, and floating-point postexceptions. When the RTE instruction executes after an access error, these continuation flags are tested. If they are set, the corresponding exception is immediately taken. The access error-stack frame contains the PC for the instruction causing the trace. It also contains the effective address needed by the exception handler for unimplemented floating-point exceptions and postexceptions.

Floating-point unit

The 68040 FPU is an on-chip functional unit that complements the IU. It is completely compatible with user object code in the 68881 and 68882 floating-point coprocessors and conforms to the IEEE 754 floating-point standard[3] via a software envelope. The FPU executes concurrently with the IU. The design goals for the FPU were compatibility and optimized double-precision performance.

The programmer's model for the 68040 FPU is identical to the one for the 68882. It consists of the following floating-point registers:

- eight 80-bit data,
- one 32-bit control,
- one 32-bit status, and
- one 32-bit instruction address.

The FPU supports a number of data formats:

- byte integer, (8 bits),
- word integer (16 bits),
- long-word integer (32 bits),
- single-precision binary real (32 bits),
- double-precision binary real (64 bits),
- extended-precision binary real (80 bits), and
- packed binary coded decimal (a software envelope handles this format).

The 68040 FPU implements a fundamental subset of

Table 2.
The 68040 FPU subset of 68882 instructions.

Mnemonic	Instruction
FADD	Add
FCMP	Compare
FDIV	Divide
FMUL	Multiply
FSUB	Subtract
FABS	Absolute Value
FSQRT	Square Root
FNEG	Negate
FMOVE	Move
FTST	Test
FBcc	Branch on Condition
FDBcc	Decrement and Branch on Condition
FScc	Set on Condition
FTRAPcc	Trap on Condition
FMOVEM	Move Multiple Registers
FSAVE	Save Context
FRESTORE	Restore Context

the 68882 instructions in hardware (see Table 2). All other 68882 instructions are emulated in software.

To change rounding modes or rounding precisions, 68882 software in the past had to read the control register, modify the appropriate bits, and place this value into the control register. Users and our own compiler designers requested that rounding precision control be implemented without the performance penalty of manipulating a control register. To this end, we added new variants to the basic floating-point instructions. These variants execute in the same way as the original instructions, except that the rounding precision specified by the control register is ignored and either forced to round to single- or double-precision, depending on the variant. For example, FADD creates a Floating-Point Add instruction, FSADD forces the result to be rounded to single precision, and FDADD forces the result to be rounded to double precision.

The 68040 does not impose an execution-time penalty for any rounding precisions or rounding modes. All calculations are always carried out to internal extended precision, with the result rounded to the destination precision.

Consistent with the user object-code compatibility goal, the FPU is truly separate from the IU. The exception model of user signal handlers already written for the 68882 coprocessor assumes this separation since the 68882 is separate from the CPU.

The FPU acts as a slave to the IU. The IU passes data to the FPU and queries for results. The FPU does not interrupt or make requests of the IU. This procedure exactly mimics the manner in which a 68882 communicates with its parent CPU. However, the 68040 FPU has the advantage of being on the same chip as its master. We used this advantage to let the FPU monitor the early stages of the IU instruction pipeline. By the time the IU is ready to pass or collect data, the FPU has predecoded the request and is ready to accept or send the data. Thus, we have minimized stalls in the IU caused by the FPU. The FPU accepts data and releases the IU even though the FPU may not be ready to start the accepted instruction (for example, register dependencies).

Floating-point pipeline

The FPU consists of a three-stage pipeline: the floating-point conversion unit (FCU), execution unit (FEU), and normalization unit (FNU). (See Figure 5.) The FCU is the only stage that communicates with the IU. It retrieves the operands, tags them (as normalized, denormalized, zero, infinity, or not-a-number), and converts them to extended precision. The FCU also performs an initial add or subtract operation involving the exponents to prepare for a possible alignment shift in the FEU. The FEU contains a mantissa 67-bit arithmetic unit (AU), a 67-bit barrel shifter, and a 64-bit × 8-bit multiplier array. The FEU performs the required calculations to produce an unrounded intermediate result. In fact, the FEU performs all of the fundamental floating-point algorithms. The FNU normalizes and rounds the result to the desired precision and also checks for exceptional conditions. The FNU then writes the result into a floating-point data register in the register file over a dedicated write-back bus (not shown).

With a 32-bit data port to the IU, transferring double-precision operands between the IU and FPU requires two clock cycles, while extended-precision transfers require three. The first stage of the FPU requires one additional cycle to detect IEEE 754 default cases (such as a floating-point multiplication of zero and infinity). For example, a dyadic instruction that has one operand in a floating-point register and the other operand from memory in double-precision format would occupy the first stage (the FCU) of the FPU for three clock cycles. However, once the double-precision data is received by the FCU, it immediately passes to the FEU so that the first clock cycle of execution in the FEU overlaps the third cycle of execution in the FCU. Although this overlap does not affect the fully pipelined execution time, it reduces the pipeline latency of an instruction by one clock cycle.

Fully pipelined Floating-Point Add and Subtract

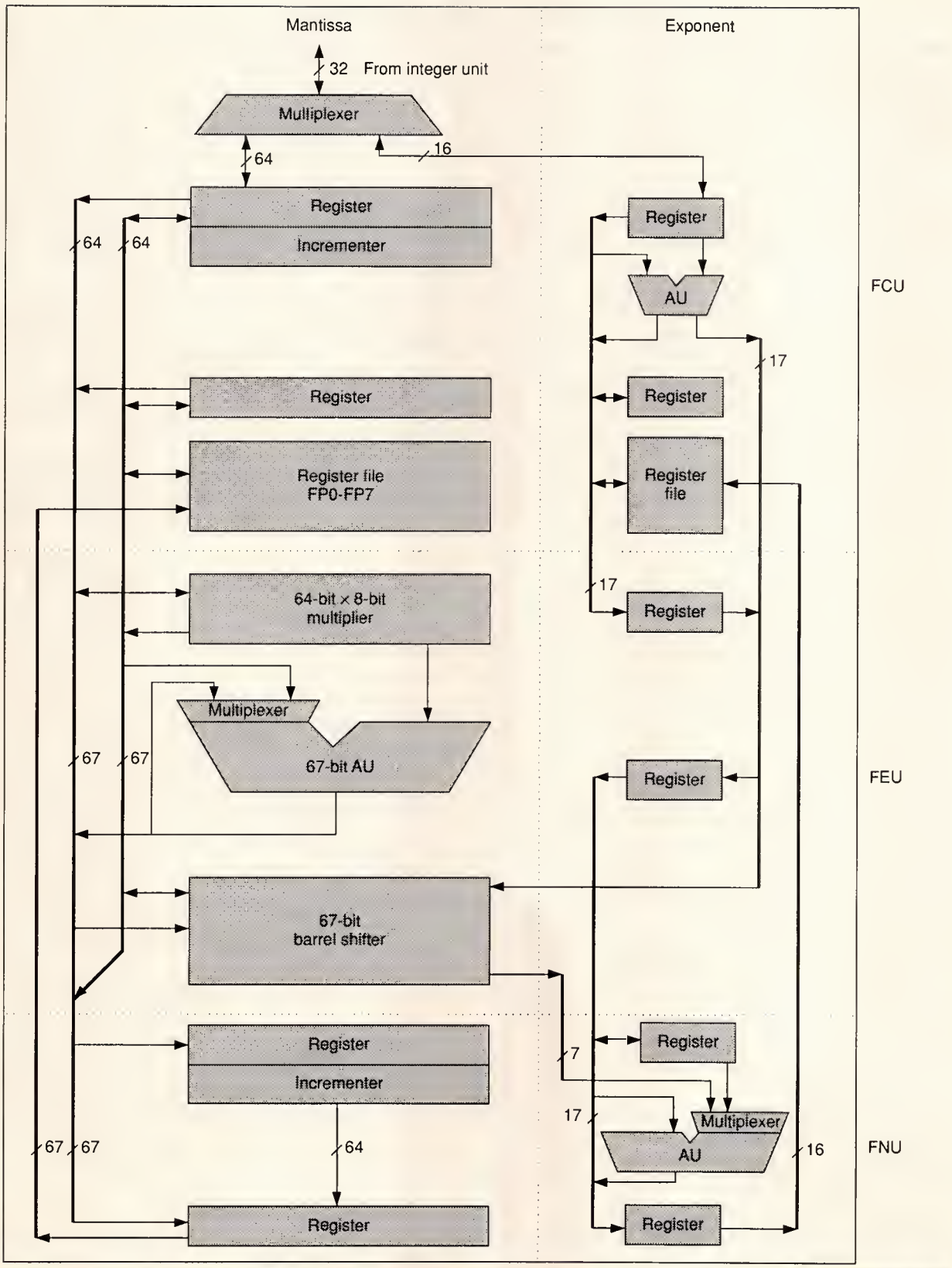


Figure 5. Floating-point unit of the 68040.

Table 3.
Floating-point instruction execution for the 68040 in clock cycles.

Instruction mnemonic	Source	Destination	FPU detail			Fully pipelined, 25 MHz	
			FCU	FEU	FNU	68040	68882
FMOVE	Register	Register	2	0	0	2	21
FMOVE.D	Memory	Register	3	0	0	3	40
FMOVE.D	Register	Memory	3	0	0	3	44
FADD	Register	Register	2 (3)	3	2 (3)	3	21
FSUB	Register	Register	2 (3)	3	2 (3)	3	21
FMUL	Register	Register	2 (3)	5	2 (3)	5	76
FDIV	Register	Register	2 (3)	38	2 (3)	38	108
FSQRT	Register	Register	2 (3)	103	2 (3)	103	110
FADD.D	Memory	Register	2 (3)	3	2 (3)	3	75
FSUB.D	Memory	Register	2 (3)	3	2 (3)	3	75
FMUL.D	Memory	Register	2 (3)	5	2 (3)	5	95
FDIV.D	Memory	Register	2 (3)	38	2 (3)	38	127
FSQRT.D	Memory	Register	2 (3)	103	2 (3)	103	129

instructions execute in three clock cycles with double-precision operands. The FEU executes these instructions using the 67-bit barrel shifter to align the operands, the 67-bit AU to add the operands, and the same 67-bit barrel shifter for postnormalization—all within three clock cycles. This performance equates to over 8 million floating-point operations per second on double-precision operands (if one assumes hits in the data cache at 25 MHz).

A 64-bit × 8-bit hardware multiplier array in the FEU handles Floating-Point Multiply instructions. Performing a 64-bit × 64-bit, extended-precision, mantissa multiply operation requires eight passes through the array. During these passes, the outputs of the array are pipelined to the inputs (with the appropriate shift). We simply clock the array eight times to produce the result sums and carries. Since the multiplier array can be clocked twice per primary clock cycle, it takes four cycles to produce sums and carries. One more cycle is required to add these sums and carries to produce the final result. Therefore, fully pipelined FMUL instructions run five cycles each.

The Floating-Point Divide instruction uses a non-restoring shift-and-subtract algorithm to produce results. Two of the 66 quotient bits are produced every clock cycle (for a total of 33 clock cycles). Setup for the division loop requires two clock cycles. The remainder restore operation consumes up to three cycles. Thus, fully pipelined FDIV instructions run 38 cycles each.

Table 3 lists selected instruction timings for the FPU. The FCU passes data to the FEU as soon as it is available (two clock cycles for double precision). Similarly, results produced by the FNU are also available early (within two clock cycles). These overlaps help reduce instruction latency. Clock cycle times in parentheses are the total times a given stage is occupied by these operations. The fully pipelined instruction times assume optimized addressing modes and hits in the cache. Instruction extension .D indicates double precision.

Figure 6 shows a sample instruction sequence in the floating-point pipeline. Single-cycle integer instructions can execute in the empty slots of the IU without increasing the execution time. Floating-point results are available after the second cycle of execution in the FNU even though it is busy for three clock cycles.

Note that instructions not requiring the use of the mantissa adder, barrel shifter, or multiplier do not pass to the FEU. They execute completely in the FCU. Floating-Point Move, Absolute, and Negative instructions (FMOVE, FABS, and FNEG) execute entirely in the FCU. Also, the FCU alone handles several IEEE 754 default cases of other instructions (such as the floating-point add operation of zero plus zero). The consequence of this approach is that floating-point instructions can execute and complete in an out-of-order fashion for program sequences. This procedure is allowed as long as there are no register dependencies. The floating-point data registers are dually ported and fully scoreboarded to provide hardware protection for register dependencies during out-of-order instruction execution.

Exceptions

The programmer need not be concerned with the fact that floating-point instructions can complete nonse-

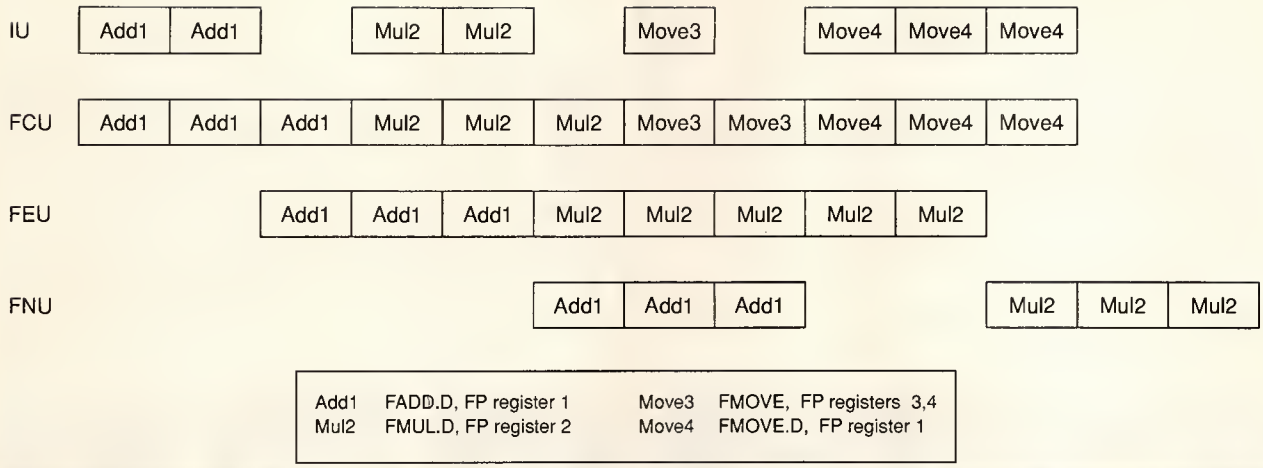


Figure 6. FPU instruction-flow diagram.

quentially. From the programmer's point of view, no reordering of instructions occurs. The system reports exceptions in the order that the programmer would expect, even when multiple exception-causing instructions execute out of order. This factor is important since user signal handlers exist for 68882 floating-point exceptions. (Floating-point exception handlers "jump" to user code so that the user can decide how to handle exceptions for a specific application.)

In-order exception reporting with out-of-order execution is possible because the floating-point exception model is a continuation model that contrasts with the restart model in the IU. Having two contrasting exception models on the same chip poses no additional programming constraints. The Save Context and Restore Context instructions behave in the same manner as in the 68882. The entire internal state of the FPU can be saved on the stack frame and restored with the FSAVE and FRESTORE instructions. FPU instructions that were saved on the stack frame simply continue where they left off once they are restored.

The 68040 FPU always minimizes the amount of state dumped onto the stack by the FSAVE instruction. The save operation of an idle FPU only stacks 1 long word. Consequently, during an FSAVE instruction, the FPU flattens its pipeline (if possible) to produce an idle state frame. This action typically results in faster context switches. If any instructions in the FPU pipeline generate an exception, the system stacks a longer "busy" state frame. However, the largest amount of state information that could be saved by the 68040 FPU is less than half of that stacked by the 68882. This reduced stack-frame size improves context-switching time.

Because it is separate from the IU, the FPU does not become directly involved in IU exception or interrupt processing. Only if the IU handler decides that the FPU is needed (such as for context switching) is an FSAVE or FRESTORE instruction necessary. The result is that the FPU can continue execution while the IU services an interrupt or integer exception.

Software envelope

All of the 68882 instructions that are not implemented on chip (like elementary functions) trap to the unimplemented instruction handler, where they are emulated in software. To ease the burden on the software emulator, the 68040 FPU creates an FSAVE state frame of only 11 long words. This state frame contains the operand (or both operands for dyadic instructions) already fetched and converted to extended precision. The state frame also contains tag information along with the instruction itself. If the emulator wants to report an exceptional condition, it sets the appropriate bits in the state frame and executes an FRESTORE instruction, which allows the 68040 hardware to handle the exception.

A software emulator of all unimplemented instructions is available from Motorola. The software emulation of elementary functions produces results that are more accurate than those of the 68882. The 68882 produces results accurate to 1.0 ulp (unit in the last place), double precision. On the other hand, 68040 software emulation results are accurate to 2.5 ulp, extended precision. The emulator's results are monotonic. Execution time of the software emulation for elementary functions including all trap overhead (running on a 25-MHz 68040) is 13 to 130 percent faster than the equivalent instructions on the 68882 running at 25 MHz. A user can further reduce execution time by recompiling with a compiler that replaces the unimplemented instructions with library calls.

In the first part of a two-part series, we have introduced the 68040, described design trade-offs, provided some insights into its implementation, and detailed the integer and floating-point units. In the June 1990 *IEEE Micro*, we will discuss the memory subsystem, the external bus, chip and board testing, and design-verification methods in the 68040.

Acknowledgments

We thank all of the people who have worked on the 68040 design and have helped to make its implementation possible.

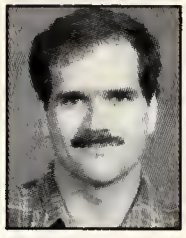
References

1. A.J. Smith, "Cache Memories," *Computing Surveys*, Vol. 14, No. 3, Sept. 1982, pp. 473-530.
2. J.K.F. Lee and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, Vol. 17, No. 1, Jan. 1984, pp. 6-22.
3. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*, IEEE Computer Society Press, Los Alamitos, Calif., 1985.

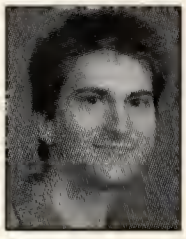
Edenfield

Gallup

Ledbetter



McGarity



Quintana

Reininger

Robin W. Edenfield is a principal staff engineer at Motorola and was primarily responsible for system design of the cache and memory management controllers for the 68040. He previously worked in the design, test, and production of microprocessor systems at several other companies. His current interests include microprocessor architecture and memory subsystems.

Edenfield received the BSEE degree from the Georgia Institute of Technology, where he was elected to Eta Kappa Nu.

Michael G. Gallup, with the microprocessor group at Motorola since 1981, is a circuit design engineer and a member of the technical staff for the high-end design group. His areas of interest include semiconductor devices, digital systems, and computer programming.

Gallup received the BSEE and MSEE degrees from the University of Nebraska at Lincoln. He is a member of the IEEE Computer and Circuits and Systems Societies.

William B. Ledbetter, Jr., is a principal staff engineer in high-end design at Motorola. He was responsible for the external bus specifications and protocol—as well as the system design of the bus controller—for the 68040. He also was a member of the architecture design team for internal memory. He previously designed graphics systems for Data General.

Ledbetter received the BSEE and MSEE degrees from Texas A&M University. He is a member of the IEEE Computer Society.

Ralph C. McGarity is a principal staff engineer and was the systems technical project leader for the 68040 project. He has spent seven years at Motorola designing 68000 family parts. His interests include microprocessor architecture design, implementation, and verification.

McGarity received the BSEE degree from Rice University and the MSEE degree from Carnegie Mellon University. He is a member of the IEEE, ACM SIGarch, Phi Beta Kappa, and Tau Beta Pi.

Eric E. Quintana has been with the microprocessor group at Motorola since 1984 as a staff engineer and is the architect for the floating-point section of the 68040. He previously worked as a design engineer on the 68020 microprocessor and later on the 68882 floating-point coprocessor.

Quintana received the BSEE degree from Texas A&M University.

Russell A. Reininger is a principal staff engineer for high-end microprocessors at Motorola. He was responsible for the system design of the integer unit in the 68040. Before joining Motorola, he worked for six years at Tracor Aerospace in Austin, Texas, where he designed radiation-hardened military computers.

Reininger received the BSEE degree from the University of Texas at Austin.

Readers can direct questions regarding this article to Ralph McGarity, Motorola Inc., M/S OE37, 6501 William Cannon Drive West, Austin, TX 78735-8598.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162 **Medium** 163 **High** 164

Department

Micro Law

Richard H. Stern
Law Offices of Richard H. Stern
1300 19th Street NW, Suite 300
Washington, DC 20036

Appropriate and inappropriate legal protection of user interfaces and screen displays

Part 5, How different forms of copyright protection interact with policy

Our inquiry in the final part of this series now returns to one question. Which proposed regime of copyright protection for screen displays best serves the public policy requirements discussed earlier? Those requirements center on encouraging progress in the software field in two ways. We need to provide incentives to innovators and investors while preserving the availability of utilitarian screen display techniques to other workers in the field.

Participants at the US Copyright Office's public hearings in September 1987 discussed the main possible regimes of copyright protection for screen displays. Basically, we have three possibilities. One regime involves conceiving the registered work as an integrated "computer program" work, which is not just the code of the computer program but also its displays, results, software specifications, and general "look and feel." The second considers the screen display as an integral part of the code of the computer program or as inherent in it (the Copyright Office policy). The third recognizes a screen display as a distinct category of copyrightable work, such as a pictorial work or an audiovisual work, separate from the code of the computer program. I evaluate each regime here.

The integrated program

Look and feel is the proverbial "pig in a poke." Proponents of this method of protecting screen displays have a broader agenda, of which screen displays are only a small part. The agenda would establish conventional copyright over every conceivable aspect of computer programs. Included would be the selection of results that the computer program will accomplish, its flowchart, program specification, metaphors and icons, command set, whatever.

Series Highlights

User interfaces for computer programs are often significant in determining how easy it is to learn to use and how efficiently users can utilize the programs. They are therefore significant factors in the commercial success of computer software. Should the law protect screen displays and other aspects of user interfaces from competitive imitation? If so, what is the best mechanism to do so?

The main question posed by legal protection of screen displays and related aspects of user interfaces concerns two issues. One is the importance of keeping the teachings of the science of human factors analysis (human-computer interaction) freely available to workers in the field. Just as important is the need to provide incentives to creators and investors in innovative screen display technology.

Some court decisions and some aggressive litigation positions of software publishers have raised legitimate industry concerns. People wonder whether copyright protections for screen displays will be used to lock up utilitarian features of screen design, thereby impoverishing the toolbox of other workers in the screen design field.

The concerns are most acute in the case of very important or necessary techniques, such as those dictated by human factors analysis. But concerns may also be appropriate about screen design conventions and emerging conventions.

A very troubling question is whether user habituation to particular user interface or screen display expedients—resulting from users' investments in self-education—should give rise to a legally protected interest.

Part 3 of this series addressed how the courts and Copyright Office deal with concerns of this type, after providing a brief copyright law tutorial. Part 4 discussed the available alternatives—mainly a sui generis industrial property system or copyright law.

Part 5 addresses some issues raised at the Copyright Office's September 1987 hearings on screen displays. These issues concern whether screen displays and other user-interface features should be assimilated to their associated computer programs for legal purposes. This installment concludes with a discussion of how the courts are likely to treat the Office's decision that these things should be amalgamated.

0272-1732/90/0200-0079$01.00 ©1990 IEEE

This view was most candidly stated in the testimony of Jack Russo, a private sector attorney. Russo urged recognition of a copyright in a single entity termed a "computer program." But this entity embraced everything involved in the program except the hardware specifications for the computer on which the program would execute.

In addition, a representative of Lotus Development Corp. stated that Lotus did not

> ... consider the works of authorship which we at Lotus create to be confined to the small portions of code which we submit with our [application for] registration, nor even to the complete code or internal structure. A program is a work of authorship in which the code represents only a portion of the creative work.

Even if there were nothing wrong with the agenda, as such, this approach would not be a legitimate way to achieve it. First, it is not the computer program law that the US Congress passed in 1980. Congress decreed copyright protection for "set[s] of statements or instructions to be used directly or indirectly in a computer to bring about a certain result." A screen display (let alone the rest of look and feel) is not a statement or an instruction, as those terms are commonly understood in computer parlance. A line of source code or object code is clearly a statement or an instruction used in a computer. But a screen display is not a statement or instruction, in the ordinary sense of those words. Indeed, a screen display is much more "a certain result" brought about by using instructions in a computer than it is like the instructions themselves.

Congress has consistently refused to extend copyright protections to results brought about by copyrightable subject matter. Thus, a book on baking pies or making churns is copyrightable, but the resulting pies and churns are uncopyrightable. Both the statutory definition of "computer program" and structure of our present copyright law are inconsistent with interpreting the term "computer program" to include screen displays or the other known elements hidden in the look-and-feel carpetbag.

Who of us knows whether it would be desirable to protect the unknown elements hidden in that carpetbag? Or whether protecting them under copyright law would hinder software progress more than it promotes it? Certainly, it is not sound statutory construction to imagine that Congress wrote a blank check in favor of software publishers. The legitimate way to bring about this agenda would be to propose it and let those affected by the proposal speak out for or against it. Then Congress should vote it up or down, or up in part and down in part.

Moreover, Congress need not opt for 75 years of injunctions, seizures, and criminal penalties for these noncode adjuncts of computer programs. Something less or different might more serve the public interest. Clearly, the integrated computer-program/look-and-feel option poses the greatest danger of preempting utilitarian aspects of the protected subject matter, and otherwise overprotecting the subject matter. As a result, the protection may unduly hinder the innovations of others and unjustifiably mulct the public.

A further problem with expanding computer program copyright to unspecified look and feel of this type is that doing so impairs business certainty. Buying a pig in a poke is an unjustified risk. Business predictability is an important social and economic value, and between two proposed legal solutions the one with more predictable results should be preferred.

The look-and-feel approach to protecting screen displays is the worst of the three copyright alternatives. Such a system would quite likely be worse than no system of protection.

Inherent code vs. separate copyrightable works

Discussion of the other two options may be shortened by considering them together. The first of these options is the "unitary copyright" and "single integral work" position of the Copyright Office. The Office followed this position just before the 1987 Softklone decision and in June 1988 officially restored it as Office policy.

The Office stated its rationale for this policy somewhat murkily: "The computer program code and screen displays are integrally related and ordinarily form a single work," and "the program code and screens are conceptually a single work." While the rationale is unclear, the result is quite clear: Software proprietors will get only one copyright registration for the computer program and screen displays of a software package.

The second option is that adopted in the Softklone case. The approach of treating screen displays as works of authorship that are independent of the computer programs (codes) that generate them is based on the approach that the courts have adopted for the screen displays of video games. (The courts have considered video games to be audiovisual works separate from the code that generates them.)

I have tried to show, earlier in this series, that the Office's June 1988 policy statement on single registrations of screen displays does not withstand facial analysis. It suffers also from practical problems—failing to meet industry and public needs. There are two principal problems, both of which have already been exposed in litigation over screen displays. One is the implausibility (or "Emperor's new clothes") problem. The other is that one copyright makes it harder to prove a case of substantial-enough taking to support a finding of copyright infringement.

Proprietors of screen displays have recognized both of these problems. Thus, whoever separately registered the screen displays of the Crosstalk program, in anticipation of the Softklone litigation, understood and anticipated the problems. Lotus anticipated the problems, too, when it unsuccessfully sought a separate copyright registration of the screen displays of its 1-2-3 spreadsheet program, before suing Mosaic Software, Inc. and Paperback Software, Inc.

Finally, Manufacturing Technologies, Inc., the plaintiff in the CAMS case (as discussed in Part 3 of this series), anticipated the problems by securing registrations of some of its screen displays in preparation for copyright infringement litigation. Moreover, the court in that case understood and discussed at least the second problem. (It decided to deal with it by creating a legal fiction that screen display/computer program copyright registrations should be interpreted

in a manner disregarding the Copyright Office's unitary copyright policy about them.)

The first problem is that screen display is not the code or the computer program, and does not look like it, any more than the audiovisual display for a video game is its code or computer program, or looks like it. The difficulty in persuading a court that a copyright registration of a computer program covers the associated screen displays is manifest. At least two courts (Softklone and CAMS) have stated their refusal to believe that a visual display on a TV monitor is the same thing as a computer program. To be sure, the Copyright Office has now told the courts that screen displays are part of computer programs, for copyright purposes. But the courts may not believe it; we have the problem of the Emperor's new clothes.

The second problem is, as the CAMS court recognized, that appropriation of screen display material may appear to be only an insubstantial taking when viewed as a fraction, which is probably only a small percentage, of a total computer program composite work. The same taking may be a large percentage or an important part of the screen display material considered by itself. Thus, conflating a computer program code and screen display tends to deprecate the substantiality and importance of an unauthorized taking of the screen display. It is far better to focus the claim of copyright infringement on the parties' screen displays—without the distraction of thousands of lines of irrelevant, nonsimilar, and uncopied code. Doing that facilitates demonstrating to the fact finder that the accused work is substantially similar to the copyrighted work.

Of course, it may be possible to overcome the difficulty by pretending ("creating the legal fiction") that one copyright registration is really two registrations. Thus, the court in the CAMS case stated that it would devise an "approach that conform[ed] to the realities of Copyright Office registration procedures." That statement meant that the court would recognize that "a computer program and its screen displays are, for copyright purposes, fundamentally distinct." The court therefore created what it referred to as the "legal fiction" that there were two separate copyright registrations, one of the code of the computer program and one of the screen displays. The court addressed them and their possible infringement separately.

But all of this is a needless burden on litigants and productive of uncertain outcomes. The problems created in enforcing a screen display claim of copyright based on the registration of a computer program weaken the benefits and incentives that the copyright system offers creators of screen displays. A weak and ineffective copyright is uncertain and costly to those involved.

If we grant that screen displays should be protected by copyright, it makes no sense to protect them ineffectually. The single copyright registration for computer programs and screen displays is ineffectual. In contrast, an independent copyright provides the strongest and most enforceable copyright for screen displays. It readily permits a finding of copyright infringement to be made on the basis of substantial similarity between the parties' screen displays, regardless of whether the underlying computer program codes are substantially similar.

Possible further advantages would flow from focusing attention on the screen display, for registration purposes. That would make it more likely that a record (and administrative procedures ensuring creation of such a record) will be developed that distinguishes the copyright owner's contribution from the public domain. By the same token, it facilitates evaluation by competitors and would-be market entrants of their risk of being held liable for copyright infringement, and of what they should do to avoid infringement. It also eases the task of courts in the event of litigation. All of this in turn promotes greater business certainty and security of expectations.

Of course, clarifying the scope of screen display copyright is the very thing that the Copyright Office said in its June 1988 policy statement is none of its concern. But that view is too narrow. Concededly, the Office lacks the technical expertise to evaluate screen display candidates for copyright registration, in the manner that the Patent and Trademark Office evaluates the subject matter of patent applications. The Copyright Office cannot make a determination of whether aspects of a screen display sought to be protected are part of the prior art or are utilitarian. But that does not mean that the Office should not structure its precatory comments in circulars and other publications, its application forms, and other aspects of its registration procedures. Publication would facilitate the courts' and the public's distinction of the protectible from the unprotectible.

It is not enough for the Copyright Office to state that it is "sympathetic" to this problem and then ignore it. Requiring separate registration of screen displays, and requiring applicants to direct to the attention of courts and the public what is claimed to be protected, should be part of the registration procedure. That major step forward would help the people with a need and right to know these things to understand the scope of a screen display copyright.

For these reasons, the most logical choice among the legal-protection alternatives for screen displays that are available to a policymaker would be to protect screen displays under existing copyright law, as the Copyright Office *sub silentio* decided. The policymaker should also approach the displays as works distinct from the computer programs to which they relate, as the Copyright Office has refused to do.

What next?

I do not mean to suggest that this supposed net benefit of a system of copyright protection for user interfaces can be realized *only* by a system in which screen displays they are separately copyrighted, independent of the computer programs to which they relate. Also, I do not mean to assert that the issue of legal protection format is more important than the issues of whether there should be protection and what should be its scope. Each of those would be overstatements.

Quite possibly, the course selected by the Office, which is intermediate between that just recommended and the worst case scenario of look and feel, may prove satisfactory. Moreover, in this imperfect world it is always pos-

sible to realize even worse results, given enough opportunity and determination.

Look and feel, which is worse than conflating code and screen displays, is not the only bad idea that was available to the Office. Another available bad idea would be for the Office simply to refuse to consider screen displays protected at all. Suppose the Office had issued a policy statement that screen displays are no more protected from competitive imitation by copyright registrations of depictions of them than New York's Triboro Bridge is protected against copying by a copyright in a picture of the bridge. Suppose further that it then refused to do anything more on the ground that screen displays, whether communicative or not, were wholly utilitarian. That would be worse.

Nonetheless, I believe it wise not to conflate screen displays with code as the Office has insisted upon doing. To do that directs attention away from issues on which attention should be focused. It also invites reaching the kind of counterproductive judgment that occurred with the highlighting technique in the Softklone case. (That case involved a screen display that was separately registered, separately considered legally, but inadequately analyzed factually.) Clarity and soundness of analysis will be furthered by addressing screen displays and user interfaces as separate from their associated codes. Unfortunately, the Office, when it could just as easily have chosen a more sound approach, almost perversely took the opposite course. It is a limited satisfaction that the Office could have done worse if it had been more ill-advised.

What the courts will now do is uncertain. On the one hand, a body of video-game case law holds the computer programs and screen displays for video games to be independent works of authorship. The Softklone decision applies that doctrine to non-video-game screen displays, which seem in principle no different from video-game displays. Indeed, the Office recognized that its present position on screen displays is inconsistent with the video-game case law and registration practice under it. The Office announced that henceforth it will refuse to register video-game computer programs and screen displays separately. It thus consciously takes a position on both video-game and other computer program screen displays in opposition to the weight of judicial precedent in the field.

The next development may therefore occur in the judicial or legislative arena. The former would seem more likely, since ordinarily Congress cannot be moved to act without showing a parade of horrible examples, such as seemingly unfair judgments of courts. Thus, a case testing the Office's policy could occur if the Softklone facts arose again. Under this hypothesis, which is essentially what occurred when Lotus tried to register the screen displays of 1-2-3, the software proprietor would again seek to register both the code and the screen displays of the computer program. But this time the Office would issue a registration only on the computer program code. In refusing to register the screen displays separately, the Office might advise the software proprietor that the Office considered the registration of the computer program code to carry with it copyright protection of any authorship in the screen displays.

Assume that the software proprietor, unlike in the Lotus 1-2-3 case, refused to acquiesce in the Office's refusal to register the screen displays separately. It could seek judicial review of the refusal to register. One way to obtain judicial review of a denial of registration is under one of the provisions of the Copyright Act. This provision permits a copyright infringement suit on a copyright that the Office has refused to register. It requires the court to decide whether the registration should have been issued; the Office must be notified and may intervene. Alternatively, the applicant may seek direct review without involving an infringer, essentially by suing the Office and thus appealing to the court from the denial of registration. In either case the question is, Should the court hold that the Office erroneously refused to issue a copyright registration on the screen display as a separate work?

One possible scenario in such a copyright infringement suit is a replay of Softklone. Let us imagine that the same plaintiff markets a new program, Hyper Crosstalk. Imagine also that the same defendant makes a new clone, Hyper Mirror, whose code is different but whose screen displays are similar; that the plaintiff sues the defendant; and that the case comes before the same judge as before. In this hypothetical case, the plaintiff has tried to register the Hyper Crosstalk screen displays and computer program separately, but the Office issues only a single computer-program registration on both things.

The court would first address infringement of the computer program copyright that the Office actually registered. It would also likely find that the set of instructions in the defendant's Hyper Mirror computer program is not at all like the set in the plaintiff's registered Hyper Crosstalk computer program. Upon so ruling, the court would be obliged to consider the legal opinion of the Copyright Office, stated in the June 1988 policy statement. There, the Office took the position that a copyright registration of a computer program includes the screen displays generated by the program. As a result, copying the screen displays infringes the copyright that the Office actually registered—regardless of whether the plaintiff's and defendant's sets of instructions in their respective computer programs are similar. Unless the judge had a change of mind about the scope of a copyright in a computer program, the ruling would be the same as before. That is, the defendant did not infringe the copyright in the plaintiff's computer program.

The court would then proceed to consider whether the Office should have issued a copyright registration on the plaintiff's screen displays. Presumably, the court's rationale would be the same in answering both of the following questions. 1) Did the defendant's screen display infringe the copyright in the registered computer program? 2) Did the Office act within its authority when it refused to register the screen displays as such? That is, either the Office was right, in which case the answer to both questions is yes (and the defendant infringed the actual copyright). Or it was wrong, in which case the answer to both questions is no (and the defendant infringed only the copyright that should have been, but was not, issued).

A court will usually give considerable deference to an agency's interpretation of the statute under which it operates. That principle has been applied to Copyright Office refusals to register and its other decisions about how to run its own operations. Thus, the Seventh Circuit has held that the Copyright Act vests "broad authority in the Register of Copyrights to fashion a workable system of registration and deposit of copyrighted works."

The deference that courts give an agency's interpretation of its statute may depend on a variety of factors. When the agency's interpretation alters the expressed intent of Congress, no amount of expertise will convince the court that the agency should be allowed to rewrite the statute. By the same token, courts are skeptical of constructions that interpret the language of a statute in a way different from its ordinary meaning. A long-standing and consistent agency interpretation, on the other hand, is given great weight, particularly if uniformly followed since enactment of the statute.

In the intermediate case, where Congress has not addressed or only ambiguously addressed the point in controversy, courts will usually defer to the agency's interpretation if it is reasonable, rather than arbitrary and capricious. They will do so even if their own interpretation might have been different. The rationale is that the agency has more expert knowledge than courts do about the matters under regulation and about the force and impact of statutory policy in different given fact situations.

Applying these principles to specific fact situations, such as the hypothetical case replaying the facts of Softklone, can be problematical. This is illustrated by the Supreme Court's recent set of fragmented opinions in *K Mart Corp.* v. *Cartier, Inc.*, concerning Customs Service regulations dealing with trademarked goods and parallel importation. The regulations interpreted a part of the Tariff Act excluding importation of trademarked goods to be inapplicable to certain fact patterns. These patterns were: a) a domestic subsidiary of a foreign parent seeks to bar importation of goods sold by the parent, b) a domestic firm seeks to bar importation of goods that it or its subsidiary made and sold abroad, and c) a domestic firm sought to bar importation of goods made abroad by its licensee.

All members of the Court agreed that the statute was ambiguous in the first case (domestic subsidiary of foreign parent) and that the agency's interpretation of the statute should be upheld. Six members of the Court appeared to accept the principle that the agency's interpretation of such an ambiguous statute must be upheld unless capricious, and voted to uphold the regulations on that theory. The other three members of the Court appeared to have accepted the agency's interpretation because they thought it was correct and also was buttressed by the deference owed to a long-standing agency interpretation.

Five members of the Court voted to uphold the agency's interpretation of the statute in the second case (goods that the complaining party made and sold aboard), and four dissented. The four dissenters saw no ambiguity, and rejected the agency's interpretation as contrary to the wording of the statute. Two of the five-member majority again relied on the principle that an agency's interpretation of an ambiguous statute must be upheld unless capricious. Three members again accepted the agency's interpretation because it was correct and was buttressed by the deference owed to a long-standing agency interpretation.

In the third case (goods made abroad by licensee), the previous four dissenters and one member of the previous majority of the Court struck the regulations down, because the agency's interpretation conflicted with the statute. The remaining four Justices, now in dissent, would have upheld the regulations as both correct and buttressed by the deference owed to a long-standing agency interpretation.

Clearly, ambiguity is in the mind of the beholder; it is little more than an excuse for an otherwise-derived conclusion on the merits. It is therefore uncertain whether a court, in a case like the hypothetical Softklone II, would rule. It might consider the Copyright Act ambiguous or clear on whether screen displays should be regarded as some sort of emanation from the copyrighted computer program code, rather than as independent works of authorship.

In part, the answer depends on how one frames the question. If one asks, Is the statute ambiguous on whether screen displays should be regarded as independent works of authorship, the answer is probably yes. Why? The statute does not address the question when it is framed in those specific terms. Hence, its silence arguably creates ambiguity. On the other hand, if one asks, Is the statute ambiguous on whether screen displays should be regarded as some sort of emanation from the copyrighted computer program, the answer is probably "no ambiguity." Why? The statute defines "computer program" as a set of instructions to a computer. A set of screen display images is not a set of instructions to a computer.

According to the particular brand of metaphysics or cant supposedly followed in the K Mart case, presence or absence of ambiguity in the statute is the touchstone for upholding or rejecting agencies' statutory interpretations. Presumably, therefore, a court that is not pleased with the Office's approach will frame the question in terms of the statutory definition of "computer program." It will see no ambiguity and reject the Office's policy statement. Such a court would rule that this definition unambiguously excludes from the scope of that term pictorial or other images from a screen display; the images are not instructions comprising a computer program. The Office's construction of "computer program" as embracing screen displays is therefore inaccurate and unreasonable.

Even if one frames the question in terms of whether screen displays, as such, are works of authorship, the statute is not wholly silent on the issue. The statute does not refer to screen displays. But it does designate as works of authorship both audiovisual works and pictorial or graphic works, and it defines them. Any screen display comprising a set of related images, therefore, should qualify as an audiovisual work. At least, it should in the light of the video-game case law interpreting what makes up an audiovisual work. An individual screen display might also be considered a pic-

torial work. Arguably, a textual screen display could also be considered a separate literary work.

Thus, although the statute does not speak of screen displays, it speaks of various copyrightable works within whose definitions screen displays comfortably fit. A court may therefore properly conclude that the Office's interpretation that screen displays, as such, are not copyrightable works is inconsistent with the legislative intent as expressed in the definitions of "audiovisual work" and other works. From that, it follows that the Office's policy alters the expressed intent of Congress and thus is arbitrary and capricious.

Other material considerations are that the Office's present interpretation is not long-standing and uniform. To the contrary, the Office has shifted position. Further, the Office's present interpretation is contrary to court of appeals and district court decisions in video-game cases. These courts said that screen displays and computer programs are distinct works of authorship, either one of which can be infringed without infringing the other. Nor does the Office have any particular expertise in the computer field to make a court feel that the agency knows any more than the court does about the impact of a decision one way or the other. Any expertise of the Office is strictly in belles-lettres.

For these reasons, the hypothetical case testing the Office's policy statement should be decided in favor of the software proprietor seeking an independent copyright in its screen displays, as well as in its computer program. But the Court's fragmentation of decision in the K Mart case shows how unpredictable litigation of this type is. Moreover, small variations in the fact pattern might well lead to different outcomes.

A November 1989 decision of the US appeals court for the DC Circuit illustrates still another judicial option—punting while wagging a finger at the Copyright Office. In *Atari Games* v. *Oman*, the court reversed a decision of the Office not to register a copyright claim on the audiovisual work aspect of a video game called Breakout. The Office considered Breakout to be just a collection of squares and circles lacking sufficient visual content for the game to be an original work of authorship. In other words, the displays were too simple and trivial for there to be copyrightable content.

The court of appeals said that it was not possible to ascertain from the Office's decision papers what standard of originality the Office was employing. It also said that that standard might be too high and that the Office might have impermissibly used a higher standard for video games than for other works. The court commented that it could not judge whether the Office had committed an error because it could not tell what the Office's rationale was. It therefore sent the case back to the Office for it to write a new decision, strongly suggesting, but not holding, that the standard of originality of authorship used this time had better be very low.

In so ruling, the court specifically rejected the Office's argument that Atari did not need a screen display copyright registration on the video game, because it was entitled to a computer program copyright registration on the computer program. The court asked, What if somebody copies the screen display without copying the computer program? There was no mention of the Copyright Office's June 1988 policy statement on screen displays.

A final comment

A number of legitimate concerns exist about the proper scope and interpretation of copyrights on screen displays. What should and should not be copyright infringement of a registered screen display? In a worst case scenario, actualization of the concerns could harm the public interest by a too-sweeping concept of infringement. On balance, however, recognizing the independent copyrightability of screen displays most likely maximizes potential public benefits and minimizes potential risks of harm to the public.

In the context of existing law, the most well-fitting, most predictable, and least harmful way to protect screens is with a separate copyright registration under the existing copyright law. Doubtless more appropriate statutory mechanisms could be devised, but copyright registration is legitimate under the existing copyright laws. It is the best immediate solution. Failure to recognize the right to copyright protection of screen displays would likely invite chaos in the interpretation of claims of copyright in computer programs. It would do no manifest violence to existing law, and it would substantially further the public interest, to treat screen displays for computer programs as being on a par with screen displays for video games and independent from their underlying codes.

We have several conflicting lines of authority at this time, in the courts and in the Copyright Office. The better authority is that which treats screen displays as independent copyrightable works. The view that a screen display is in some way subsumed within the copyright on a computer program, or is inherent in the code of the program, is unsound. The view that screen displays are part of the protectible look and feel of computer programs is even more unsound. Potentially, this view is very dangerous to the software industry and the public. I hope the view that screen displays are independently copyrightable works will ultimately prevail in the courts and Copyright Office.

But passion over detail should not dominate our thinking. It is most important that the policies discussed earlier in this series should prevail. Whatever kind of copyright is issued for screen displays or user interfaces should be interpreted to cover only purely expressive aspects of such contributions. The copyright should not be interpreted to cover utilitarian, functional aspects (that is, those due to human factors analysis). To the extent that this interpretation of screen display copyrights prevails, it will most promote the progress of software technology and maximize the benefits of the copyright system to the public.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 171 **Medium** 172 **High** 173

Micro Standards

Carl Warren
McDonnell-Douglas
(714) 896-3311, ext. 0669
MCI mail: 310-9380; Compmail+ c.warren

A busy year ahead

This is the year you can become involved with standards. Give it serious consideration. Clearly the 1990s will mark another era of technology growth. The need for standards, especially those at the interface level, will be even more important than ever.

With this in mind, I offer some key dates for consideration. During every odd-numbered month on the second Monday, the Microcomputer Standards Committee (MSC) meets at the Los Gatos Lodge in Los Gatos, California. The remaining schedule for 1990 is March 12, May 14, July 9, September 10, and November 12. You're invited to participate in these meetings and eventually take an active role. The get-together begins at 6 p.m. with an informal cocktail hour, then the dinner meeting at 7 p.m. Dinner normally costs $20. If you do plan to come, please inform either the MSC secretary, Fritz Whittington, (214) 575-6470, or treasurer, Bob Davis, (408) 353-2706.

The MSC meetings don't appeal to the faint of heart. Be prepared for interesting debate and, in some cases, downright anger. The standards business can be a touchy subject. Besides addressing standards issues, the MSC also provides a forum for presenting an interesting technology nuance, pointing out areas that need standardization, and occasionally showing a new product.

A metric measurement

For example, the November 13, 1989, MSC meeting heard a presentation about the merits of hard versus soft metric by Hans Karlsson, chair of the Metric Mechanical Study Group. Karlsson's presentation was timely in light of the impending 1992 date for the consolidation of the European Common Market. Although metric hasn't found all that much favor in the US, it is rapidly becoming an important factor for connectors and systems that will be sold throughout Europe. Moreover, due to the events of the last month of 1989, I expect large new markets to open up in Europe. I believe metric will become the measurement of choice.

Karlsson's presentation focused on hard metric, Futurebus+ board sizes with 0.5-, 1.0-, 1.5-, 2.0-, and 2.5-millimeter connector grids. He believes this approach will ensure acceptance with all European standards and manufacturer bodies, plus offer technology advantages for future developments. Karlsson argued that his study group's proposal would ensure a common solution for bus systems, computers, telecommunications, automation, and power engineering. I felt his power argument was unclear since advanced systems will probably use a more sophisticated approach than the type he alluded to.

The notion of hard versus soft metric provided key interest. The soft metric standard describes those parts that are standardized on the English system and have a metric equivalent based on a conversion factor. Hard metric, on the other hand, is defined in metric and needs no conversion or conversion tolerance.

The argument in favor of soft metric is that it is good enough for most designs based on acceptable tolerances. However, Apple Computer's Mike Teener pointed out that this assumption doesn't always hold. When a choice has to be made, Apple tends to use hard metric for a higher assurance of accuracy.

A supporting standard

Karlsson's presentation wasn't just of passing interest. Rather, he submitted a Project Authorization Request (PAR) to the MSC for an IEEE standard for a Metric Equipment Practice. The MSC agreed to the PAR, and the work will continue.

Karlsson is looking for interested par-

0272-1732/90/0200-0085$1.00 © 1990 IEEE

ties, so you might want to contact him at Ericsson Telecom, VB/ETX/TX/FG, S126 25 Stockholm, Sweden; phone 46 (8) 719-6037; fax 45 (8) 719-2900.

A matter of conscience

A coworker of mine recently asked me how one works within a standards body. He saw it as a round table to share ideas that end up being bogged down by bickering. His insight didn't fall much short of the truth. But he was comparing the process to so-called standards efforts found in large companies, a comparison that I'll address later. Although I'm a member of ANSI (American National Standards Institute) and other standards organizations, I'll confine myself to IEEE work.

Information Survey

An area that concerns the IEEE is how best to distribute information. Mail and other methods of shipping are expensive, and we are developing mounds of hard-to-handle paper. The IEEE Computer Society is considering methods of electronic distribution. But to make the right choices they need to hear from you. With this in mind, please join in the minisurvey that follows.

Name:_____

Organization: _____

1. What type of desktop computer do you use?_____

2. Would you buy a CD-ROM of IEEE publications?_____

3. Would you pay for an on-line subscription of IEEE publications? ____

4. Would your company serve as an electronic node for distributing working-group papers?_____
If yes, let me know the type of system, how many on-line users it can accommodate, and any restrictions.

Please fax to Carl Warren, McDonnell-Douglas, M/S A95-J845-17/6; ext. 0669; fax (714) 896-5034.

An IEEE working group consists of individuals interested in a given area. For example, people with an interest in communications have migrated toward the 802.X working groups, while system engineers and designers have tended toward the bus working groups that are handled by the MSC.

Fundamentally, the IEEE offers something for everyone from hardware to software—even to developing standard glossaries. Regardless of your interest, a fundamental underlying principle exists: You vote your technical conscience. What you end up with is a chance to prove your technical mettle while helping to create a viable industry standard.

Not so perfect

Unfortunately, this world doesn't approach perfection, and none of us can do standards work out of the goodness of our hearts. We must have some support. Our support comes from the organizations we work for. Thus a specific directive from the boss may taint our technical consciences.

The more important the standard looks (that is, the greater the chance of producing revenue), the greater the influence of the supporting companies. For example, consider the VMEbus. It's not necessarily the best bus in the world, nor the worst. It did hold promise as a base for building higher performance systems, thus providing what looked like solid revenue.

Futurebus+ is now drawing the same swell of interest. Obviously, manufacturers who have a strong interest in it are eyeing it as a strong base for future systems.

Just because a large company decides to influence the standards process doesn't guarantee that it wins. This influence can and does disturb the standards process. Again, the Multibus and VMEbus standards serve as good examples since fragmented committees developed them and the decisions didn't necessarily reflect the best consensus. Standards currently under development like Futurebus+ don't seem to be falling prey to this problem. Rather, consensus appears to be the name of the game.

A factor that does hurt the standards effort is minimal interest. A standard can become fragmented—changed—if members don't participate.

So what is my point? If you plan to work on a standard, do so. Be prepared to vote your best technical conscience even if it's paid for by your boss. But be there. Remember all your work can go down the drain if you lose interest. Furthermore, the final standard may have little or no value.

Things to work on

Those of you who are interested in doing standards work might want to consider joining the following working or study groups:

- P1394, Serial Bus (common to 896, 1196, 1296, 1396, and 1596). Contact Michael Teener; (408) 974-3521.
- P1596, Scalable Coherent Interface (SCI). SCI may end up being the standard of the decade. Contact Dave Gustavson; (415) 926-2863; fax (415) 323-3626.
- Fiber Optics Standards Issues. This study group looks at fiber optics as a universal transport medium. This year we expect to petition the MSC for a PAR on standardizing power-conversion interchange systems. Contact Carl Warren; (714) 896-3311, ext. 0669; fax (714) 896-5034.

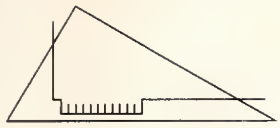
This list only samples some of the work being done. You might be working on something that would assist standards developments. Or you may be working in an area that could benefit from a standard. If you let me know, I will present it to the MSC as a possible area of consideration.

In the next issue, I will provide an update on the SCI project.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 189 **Medium** 190 **High** 191



On the Edge

Carl Warren
McDonnell-Douglas
(714) 896-3311, ext. 0669
MCI mail: 310-9380; Compmail+ c. warren

Wire-to-wire interaction

Readers do have an interest in how buses work! The number of phone calls I have received since the debut of On the Edge underscores this fact. As I suspected, reader interest centers on deterministic linear-type buses such as the Mil-Std 1553. Consequently, I've decided to delve into the operation of the serial, or wire, bus. In the next issue, I'll start with SPICE and explain how you can build a high-fidelity bus structure.

More information, please

Virtually every caller asked the same question: "How can I learn more about buses in general?" The obvious answer: Keep reading *IEEE Micro*. And I hope you all do. However, the answer I gave callers is "Go find a good antenna engineer or technician." A bus is a transmission medium, which I mentioned in the last column. Therefore, a bus tends to act like an antenna, and the same rules apply, more or less. Basically, a transmission line is a transmission line regardless of how you implement it. You don't change the fundamental physics of operation.

The antenna aspects of a bus, especially a serial bus, become evident when the bus helps control something at a distance. Consider a three-wire, RS-232C serial bus. This prevalent standard helps devices control and communicate with everything from printers to modems. It even supports the creation of low-level LANs (local area networks).

The system generally works well when the distances from the controller to the receiver are short. However, errors begin occurring at about 1,000 feet. Electrical interference aside, what causes the problems? The answer isn't simple. To find it, we need to inspect the system. Do you remember your physics professor's suggestion that you break down complex systems into small, easily understood elements? Let's do the same with our serial wire.

Smaller is better

For our purposes, it doesn't matter what type of copper-wire serial bus we inspect. The physics stays the same. Let's consider the length of the wire. As it increases, so do the impedance and capacitance. Depending upon how the wire is routed, we should consider inductive effects as well. Further, the configuration of the wire plays a big role in the system analysis. I happen to like to use flat cable for RS-232C runs. It's convenient, lays flat under floor mats, and can be routed in and out of a chassis or rack. The RS-232C works well for runs of no greater than 20 feet. Longer lengths produce dicey reliability.

A long run may call for a twisted/shielded-pair cable. Although a serial bus like the RS-232C can operate well with three wires (transmit, receive, and chassis ground), the careful engineer would want the secondary channels and system grounds as well. Even under these conditions, the impedance increases with the length. The frequency—how fast electrons proceed down the wire—also figures into the impedance equation. The higher the frequency, the greater the impedance becomes in relationship to the length. Thus

0272-1732/90/0200-0087$1.00 © 1990 IEEE

the derivative of frequency yields a characteristic impedance of the line.

The nice thing about wires is that we can characterize them and "determine" their operation according to various parameters.

Capacitance and inductance will eventually play an important role in our bus analysis. For now it's sufficient to know that any time wires lay in proximity to other wires or conducting elements, we must consider capacitive and inductive effects. Running your dot matrix printer may not present much of a problem. But when you build a high-performance airframe or space station, you have to consider everything. A 0.1-microsecond delay or spike caused by an inductive effect may indicate a major fault or failure.

Deterministic twisted pair

To focus the discussion, let's consider the Mil-Std 1553 bus. (Although revisions *A* and *B* exist,[1] let's analyze the basic bus architecture.)

The Mil-Std 1553 uses a mouthful of terms to describe the bus: aircraft, internal, time-division, command/response, multiplexer, and data. The 1553 most often appears in aircraft—the standard specifically states that the 1553 is for DoD (US Department of Defense) aircraft and then only for internal transmission of signals. The time-division multiplexer (TDM) method takes information from several signal sources at many sampling rates staggered in time. This composite train transmits over a single communication medium. In the case of the 1553, the medium consists of a twisted/shielded pair of copper wires. I will discuss the TDM function of the bus in future columns since it plays a major role in implementation and analysis of the bus.

Finally, we have command/response. Essentially the bus always listens but reacts only when spoken to. We will discuss the attendant matters of protocol and control in a later issue as well.

You might wonder why anyone would use a bus like the 1553, especially since it's been around since 1973. Several answers apply. It works, users support it, and it meets the need for most airframe applications (an important consideration when you choose buses for a design).

The 1553 also requires minimal space. A single cable is better (for space and failure considerations) than a dozen, especially in aircraft. Again, choosing the right bus for the application clearly improves the performance and reliability of the system. You must consider single-points-of-failure issues, however. (Should any of you be doing risk or failure analysis, *IEEE Micro* would like to see your input on a single-points-of-failure design.)

Not so ideal

The 1553 might sound like an ideal bus, but it's not. A serial bus only sends single bits one at a time. Obviously this limitation has some impact on the maximum operating characteristic.

The 1553B bus operates at I megahertz, which is sufficient for sensors and effectors on aircraft. The point is, trade-offs exist. A serial copper-wire bus can't meet all of the needs of today's processors. We might want to consider adding a few more lines. These lines would make the bus more robust, but would also add to its complexity.

The important decisions regarding a serial bus include what type of cable to use. Remember our RS-232C example. A flat cable worked for some implementations, but it wasn't ideal for everything. As soon as I wanted to extend the serial link into a LAN—connecting several printers and PCs—I needed a shielded wire.

Why the shield?

The shield does a number of things. A resistance network usually holds the shield itself—which is normally a braided coaxial sleeve—at reference level. The best operation reference is earth ground—zero potential. You probably guessed this isn't possible with aircraft. The ground is in reference to the airframe, which is at some differential voltage above and below ground. A DC-10, for example, has a ± DC level as the ground reference. But we can call the reference level ground. The shield provides a needed path for unwanted signals. It also establishes an internal reference to the wires so the signal limits can be determined.

Most connectors use the shield. The braid on the coaxial cable is soldered to the connector body, which brings the shield to chassis ground, or equal electrical potential. The chassis may or may not be a reference or ground potential, which can cause some interesting interaction in the bus line.

In the next issue I'll look at the interaction of the wires. I'll explain divergence and curl and how to make a signal degrade to zero without really trying.

[If you would like to pursue literature on the subjects addressed here, you can obtain most military standard documents and reference works from Global Engineering Documents, 2625 Hickory St., Santa Ana, CA 92707; (714) 540-9870, (800) 854-7179. Ask for the Mil-Std 1553A/B DoD Mil Spec document.]

References

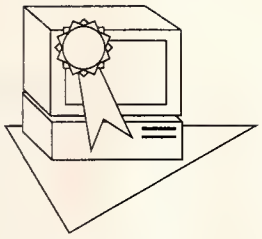
1. *Mil-Std 1553 A & B Designer's Guide*, ILC Data Device Corp., Bohemia, N.Y., 1987.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 206 **Medium** 207 **High** 208

Department



New Products

Marlin H. Mickle
University of Pittsburgh

Send announcements of new microcomputer and microprocessor products, and products for review, to Managing Editor, IEEE Micro, *PO Box 3014, Los Alamitos, CA 90720-1264.*

Neuralware

Simulate neural networks

Parallel computers modeled after the human brain can be designed, built, trained, tested, and run with the Brain Maker simulator. Brain Maker reads data from Lotus, dBase, Quattro, and Excel files and supports graphics post-processing of network results. **California Scientific Software; $195 or V1.5 upgrades for $95.**

Reader Service Number 10

Desktops simulate human vision

Specially designed VLSI chips allow a board-level processor to simulate human vision. According to the company, its Sharp Image Processing System supports varied personal computer desktop applications at 700-MIPS speeds. The chip set extracts the essentials of an image (boundaries, corners, vertical, horizontal and diagonal lines) and translates these to a circuit board.

A standard board unit includes six chips: one PIP, four PECs, and one FED. Operating at a 40-ns/pixel rate, the system's PIP chip processes a PC display screen in 1/60th of a second. The PEC devices next extract and compact higher level information at the same speed. The FED chip performs consecutive, fine-line, and label processing while functioning as the board module's CPU. **Sharp Corporation of Japan.**

Reader Service Number 11

ASIC design

IC designs verified

A family of IC design verification products called Check Mate access Texas Instrument proprietary technology for hierarchical cell design and reuse. CMOS, bipolar, and BiCMOS IC design rules can be checked and validated to ensure that an IC conforms to an intended schematic. Check Mate checks a cell's internal structure once and maintains a copy in its database so only cell perimeters need to be checked with each repeated instance of the cell. The Apollo and Sun platform-based Check Mate is scheduled for second-quarter 1990 availability. **Silicon Compiler Systems.**

Reader Service Number 12

Synthesize A/D circuits, layouts

A suite of circuit synthesis and layout synthesis tools for analog and mixed-signal (analog and digital) IC design helps users improve ICs and ASICs. With the Sun 3 or Apollo DN4500 workstation-based Explorer Autofilter and Explorer Autolinear, IC designers can create accurate schematic, simulation, and layout views of desired analog functions. They do so by specifying desired performance levels across all key parameters and accessing the software's embedded expert knowledge and numeric optimization algorithms.

Explorer Lsim simulates designs with tight synchronization and coupling between analog and digital portions of a chip. A block router called Explorer Autoroute includes features that take into account key mixed-signal design issues such as noise isolation. The router provides added control over signal widths, routing paths, and priorities.

Caeco Designer, an advanced IC layout design system, contains all-angle polygon and object-oriented layout editing features for analog design. **Silicon Compiler Systems; from $40,000.**

Autofilter **Reader Service Number 13**
Autolinear **Reader Service Number 14**
Lsim **Reader Service Number 15**
Autoroute **Reader Service Number 16**
Caeco Designer
Reader Service Number 17

IEEE/JTAG ASIC kit tests PCBs

The Boundary Scan Kit lets designers test company 1.0-, 1.5-, and 2.0-micrometer ASICs once they are placed on a printed circuit board. Eleven connectable cells in the kit add boundary scan testing to ASIC designs.

The kit supports the IEEE P1149.1 standard subset/Joint Test Action Group Boundary Scan Architecture test protocol version 2.0, which standardizes ASIC and IC test interoperability. A test library with cell schematics, an application note, and a test program template complete the kit. **VLSI Technology, Inc.**

Reader Service Number 18

i486-based products

i486 portable weighs 20 pounds

Driven by the 25-MHz i486 CPU, the P.A.C. 486-25 portable computer provides a VGA-compatible plasma display option that can backward emulate all lower display standards such as EGA, CGA, and Hercules. The display produces bright red images with 16 levels of gray scale on a black background.

The 486-25 can drive external color monitors concurrently and contains three full-size, full-height, 16-bit, AT-compatible internal expansion slots for add-in cards. A standard configuration includes 2 Mbytes of RAM; a 100-Mbyte hard disk; and a 1.2-Mbyte, 5.25-inch floppy-disk drive. **Dolch Computer Systems; $12,995.**

Reader Service Number 19

The display P.A.C. 486-25 PC produces bright red images with 16 levels of gray scale on a black background.

Sharing 386/i486 systems

PC Share/vm lets PC users share programs, data, and access to 386, 386SX, and i486 system resources. In addition, they can take advantage of the processors' protected-mode features. Each of 17 maximum users can receive 640 Kbytes of system memory and can access up to 32 Mbytes of EMS memory.

A four-user package with the PC Share/vm adapter supports four Hercules monographic or colorgraphic workstations. Each workstation can be located up to 200 feet away from the host computer. **Zaki Corporation; $599 per user.**

Reader Service Number 20

EISA/i486 PC introduced

The Premium 486/25TE i486-based EISA tower offers workstation speed and power formerly associated with RISCs. Targeted for multiuser and file server environments, the 25TE includes 10 EISA expansion slots and storage peripheral capacity for disk-intensive departmental database applications and high-resolution graphics.

The 25-MHz EISA architecture provides 32-bit data transfer, automatic system configuration, and bus master support for high-speed expansion boards. Four models are available, each supplied with a 5.25-inch floppy drive and 2 Mbytes of memory. **AST Research Inc.; from $10,245.**

Reader Service Number 21

OEMs offered i486 platform

The 486 Microcomputer PC platform Model 401, based on an AT, or ISA bus, offers OEMs multiuser, server, and workstation applications. As a platform, the 401 stands alone and provides 8 Mbytes of interleaved 80-ns RAM and 8 Kbytes of cache memory. The tower accommodates eight half-height, 5.25-inch peripherals and eight add-in expansion cards. The 401 can also function as a baseboard. **Intel Corp.; from $16,500, depending on configuration and volume.**

Reader Service Number 22

EISA units support i486 system

Two EISA system boards and three EISA peripheral interface cards ease I/O limitations. The MAE486-25/33 system board, based on a 25-MHz i486 system, includes 128 Kbytes of write-back external cache. The MBE486-25/33 system board features the EISA-bus interface without the external cache.

The interface cards include the DCE376, the LNE390, and the GXE020A.

A 16-MHz 80376 processor supports the DCE376 caching SCSI controller. The controller features up to 16 Mbytes of cache memory and bus master data transfer rates up to 26 Mbytes/s.

Mylex Corporation; from $5,500 (boards), $425 (cards).

MAE486-25/33
Reader Service Number 23
MBE486-25/33
Reader Service Number 24
DCE376
Reader Service Number 25
LNE390
Reader Service Number 26
GXE020A
Reader Service Number 27

Upgrade 386 system to an i486

The Premium 386SX/16 allows users to work with a 32-bit, 386-based system that offers a Cupid-32 upgrade path to i486 technology. The Cupid-32 architecture separates the processor and memory on the processor board from the I/O and BIOS on the system board. Components can then be upgraded without obsoleting the computer.

The 16-MHz 386SX/16 provides cache memory for zero-wait-state performance, seven expansion slots, five half-height drive bays, and a 16-bit AST-VGA graphics adapter. **AST Research, Inc.; from $2,695.**

Reader Service Number 28

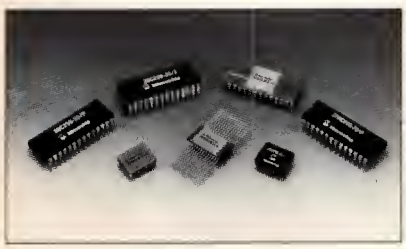
Memory upgrades

Hybrid EEPROM supports most processors

A 4-Mbit EEPROM module can be configured as 512 Kbytes × 8 bits, 256 Kbytes × 16 bits, or 128 Kbytes × 32 bits. Users choose the configuration when soldering the chip to the PC board; the grounded pin determines the bit mode. The CMOS, nonvolatile, mass-memory M4194E can be used with most 8-, 16-, or 32-bit microprocessors or systems. Designed for multiprocessor operation, the device accepts 8-bit and 16-bit or 32-bit I/O simultaneously when proper bus and interfaces are used. **White Technology, Inc.; from $2,400 (100s).**

Reader Service Number 29

Military/commercial, 90-ns EEPROMs



These 256-Kbit EEPROMs serve security, automotive, and telecommunications applications. The 28HC256 chips use a 5V power supply and have a standby current of 150 µA and an operating current of 65 mA.

A family of 256-Kbit EEPROMs includes a 90-ns device to help increase system performance. The 32K × 8-bit devices include military and commercial versions. The 28HC256 chips allow systems to execute programs at least 30-percent faster than would be possible with 120-ns devices. **Microchip Technology Inc.; from $54 (1,000s).**

Reader Service Number 30

SRAMs support 32-bit systems

The MCM6290J12 is a 12-ns, 16K × 4-bit fast static RAM that provides 64 Kbytes of memory in 33-MHz, 32-bit systems. The 1.0-micrometer, SOJ-packaged chip also has a 15-ns version. **Motorola; free samples by requesting 12NSPAK/D.**

Reader Service Number 31

DRAM promises 60-ns access times

The Future Fast family of DRAMs includes 256-Kbit and 1-Mbit chips that offer access times of 60, 70, and 80 ns. The AAA1M200P/J/Z 1-Mbit chips reside in plastic DIP, SOJ, and ZIP packages, and the 256-Kbit DRAMs in plastic DIP and PLCC packages. **NMB Technologies Incorporated.**

Reader Service Number 32

CAM aids Ethernet/FDDI

A 70-ns-cycle version of a proprietary Content Addressable Memory chip is now shipping in volume production. The Am99C10A primarily supports Ethernet and FDDI networks and bridging applications. The device functions as an address filter and performs the network look-up function. The 256-word chip compares data simultaneously to all 256 addresses in 90 ns. When a match occurs, the on-chip priority encoder generates a match word address identifying the location of the data in the CAM.

The CMOS Am99C10A resides in 28-pin, 400-mil CerDIP; 300-mil plastic DIP; and 32-pin PLCC packages. **Advanced Micro Devices, Inc.; $54.39 (100s).**

Reader Service Number 33

RAM retains data for 20 years

A 1-Mbit, CMOS, nonvolatile RAM retains data for 20 years with a UL-listed lithium power cell. The GR12882 contains automatic power-down circuitry to sense downs in 15 microseconds. Configured as 128K × 8 bits in a 32-pin DIP package, the device works with normal static RAM. **Greenwich Instruments USA; $187 (100s).**

Reader Service Number 34

EPROMs feature 100-ns access times

Sixteen-bit and 32-bit MPUs can access data in 100 ns with the 2M CMOS EPROMs. The devices feature a page-programming algorithm that lets all bits be written in about 28 seconds. The M5M27C201K 256K × 8-bit device comes in 32-pin CerDIP packages, while the M5M27C202K is packaged in a 40-pin CerDIP. **Mitsubishi Electronics; from $55 (100s).**

C201K **Reader Service Number 35**
C202K **Reader Service Number 36**

DRAM offers fast page mode

The V53C104 256K × 4-bit, CMOS dynamic RAM offers row address access times of 70, 80, 85, 100, and 120 ns. For high-bandwidth applications such as PC graphics programs, a fast page mode outputs 512 four-bit words every 55 ns for a data bandwidth of 72 MHz. **Vitelic; $14.30 (100-ns version, 100s).**

Reader Service Number 37

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number of the Reader Service Card.

Low 180 **Medium** 181 **High** 182

Advertiser/Product Index

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northern California and Pacific Northwest: Roy McDonald Assoc. Inc., 5915 Hollis St., Emeryville, CA 94608; (415) 653-2122.
Jim Olsen, P.O. Box 696, Hillsboro, OR 97123; (503) 640-2011.

Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Suite 100, Torrance, CA 90505; (213) 373-9604.

Midwest: The Kingwill Company, 4433 W. Touhy Ave., Suite 540, Lincolnwood, IL 60091; (708) 675-5755.

East Coast: Atlantic Representative Group, 349 Maple Place, Keyport, NJ 07735; (201) 739-1444.

New England: Arpin Associates, P.O. Box 6444, Holliston, MA 01746; (508) 429-8907.

Europe: Heinz J. Görgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; telex 841(17)2153310=HJG tlx d.

Southwest/Southeast: Publications Office, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

Moving?

PLEASE NOTIFY US 4 WEEKS IN ADVANCE

Name (Please Print)

New Address

City State/Country Zip

MAIL TO:
IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

ATTACH LABEL HERE

- This notice of address change will apply to all IEEE publications to which you subscribe.
- List new address above.
- If you have a question about your subscription, place label here and clip this form to your letter.

Letters

"One person's cup of tea is ..."

To the Editor:

I've been reading *IEEE Micro* for two years, since I discovered it suits my needs. I prefer precise information given in clear form and of a certain abstraction level. I used to find it inside *Micro* issues—so I kept quiet. But I'm deeply shocked and really disgusted because of "A Fixed-Point DSP for Graphics Engines" by Matthew Johnson (August 1989).

This article addresses implementation details of a very simple graphic system, chaotically mixed with a 3D graphics glossary, introduction to matrix transformation, binary arithmetic, and ADSP-2100 tutorial description. The author wastes the valuable space giving explanations of an "interesting" indexed jump table and detailed schematics of a joystick interface, decoder, etc. Such pictures should rather appear in a do-it-yourself magazine.

I haven't found a single fresh idea in Johnson's paper. In my opinion you lost 15 pages of the August issue.

Jerzy Chrzaszcz
Warsaw, Poland

The author replies:

The objective in writing the article was to close the gap between theoretical abstractions and reality by providing the necessary details for readers who intend to actually build hardware. The "fresh idea" presented in the article is that real-world and efficient implementations of generic graphics applications are now practical with a compact hardware design using the technology provided by devices such as the ADSP-2100.

Chrzaszcz appears to resent the presentation of background principals and implementation details important to a coherent and meaningful article. The purpose of providing this material was to familiarize the reader with various trade-offs and establish the foundation upon which relative performance measures could be discussed. If these specifics were left out, the article would have been too abstract to those who may not be as well versed as Chrzaszcz in the graphics area.

The choice of the DSP was based upon specific features such as zero-overhead conditional jumps and automatic circular buffer data addressing, dual data memory interface, etc., all of which bear great significance to the serious designer. Important hardware considerations such as these certainly warrant mention in any article.

The simplicity of the graphics theory presented and the hardware interface, such as the joystick, serve to demonstrate by example that a powerful DSP can perform very useful, high-performance tasks based on simple hardware.

I invite other readers to review the merit of this article. I'm sure they will find that the performance of the ADSP-2100 as a general-purpose DSP is superior in this and other applications. Performance is a key reason why Atari Games chose our system as the graphics engine for its state-of-the-art Hard Drivin' video arcade game.

Matthew J. Johnson
Norwood, Mass.

Reader Service Number 1

continued from p. 9

ware loads into the system at electronic speed, 300,000 times faster than applications accessed from a hard-disk drive.

The ROM also holds a simple memo and note-taking application, a four-function calculator, a schedule organizer, an address book, a system for accessing data by modem, and a file-transfer program employing Virtual Network Architecture. The latter provides serial-port access to a desktop PC via direct-cable connection at up to 115,200 baud, or via modem at transmission rates up to 19,200 bps. The file manager operates from a menu-driven interface. This menu and other menus are driven by the Systems Application Interface, a pulldown menuing system and windowslike environment for software applications used in the built-in applications.

The Poqet PC is not limited, however, to built-in software. It will run any MS-DOS applications software designed to run on a 512-Kbyte IBM-compatible computer. It does this by loading the application into the system RAM via a cable link from a desktop PC or, on the road, from the Poqet floppy-disk drive.

Still, it would be more convenient, especially when traveling, for users to have the software in a ROM card. To this end, publishers of 12 application packages have so far agreed to provide their software in this format. And, Poqet promises more application packages will be announced in the next few months. Moreover, these software companies plan to adapt their programs to take advantage of the benefits of operating from ROM—it's much faster—instead of from disk.

The application programs on which Poqet has reached agreements include Lotus Development Corp.'s 1-2-3 release 2.2, Agenda, Metro, and Express; WordPerfect Corp.'s WordPerfect Executive and WordPerfect 5.0; Xyquest's XyWrite III Plus word-processing program; and several others.

Human-size interface

The size of human beings being pretty well set, a display screen and the keyboard have to fit this given. Portable users want an 80-character × 25-line display with resolution comparable to what they are accustomed to on their desktop computers. They want ergonomic features such as contrast control, tilt-and-swivel capability, and black characters on a silver background. They want keys about as big as their fingers with some tactile feedback.

The Poqet PC satisfies these desires. The screen size measures 8 inches diagonally, or about 7.4 inches wide and 3.1 inches high, only a little smaller than the original Macintosh 9-inch diagonal screen. The screen aspect ratio is 2.35 to 1, better than many laptops that have short displays with a 3-to-1 aspect ratio. The screen resolution is 640 × 200 pixels. On this relatively small screen the pixels are closely spaced, producing sharp characters and graphics.

Poqet uses a proprietary, high-contrast, double supertwist, reflective liquid-crystal display, driven by an application-specific chip it designed. Since display screens are a large power user, the design team developed circuitry that reduced the display's power consumption while at the same time improving its readability.

The 77-key, QWERTY keyboard, of course, has to fit within the unit's overall dimensions, so Poqet provides a keyboard 80 percent the size of standard keyboards. The design team conducted experiments with users on variations of design, experimenting with different degrees of travel (up and down motion), tilt, and separation between keys. Spacing and travel are somewhat reduced from standard, but experience indicates that most users adapt and touchtype.

Packing in the electronics

Once upon a time the electronic circuitry to perform all these functions would have occupied a refrigerator-size box. But we have become so accustomed to packing the actual circuitry into a small space that we no longer give it much thought. Nevertheless, the design team did struggle to pack all the electronic components of the Poqet PC into this small box.

With the exception of the screen and its associated electronics, the components reside on a single, six-layer printed circuit board, using a combination of surface-mount and chip-on-board technology. The components include the RAM and ROM chips, the low-power 8088 microprocessor (7 MHz), two ASIC chips, the bus, RS-232C port circuitry, memory-card drive interface, and system and clock circuits.

Because of all the innovations developed on this project, Poqet has filed 15 patent applications that list 54 independent claims. In addition to battery-life power management, they cover display techniques, power management for the microprocessor and internal clocks, and the system software that optimizes the performance of the memory cards.

For real users

The small and lightweight Poqet PC is not a toy—as its $1,995 price drives home. It is aimed at professionals and managers. These people may already have a desktop personal computer. But they are away from their offices a few days a week, leaving behind their computerized files, documents, and reference information.

"Business people and field professionals no longer have to leave this information in their computers in the office," Stav Prodromou, president and chief executive officer of the company said. "They can take their essential files and programs with them to a meeting down the hall, or anywhere around the world, with none of the limitations imposed by the bulkiness and short battery life of other portable computers."

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 183 **Medium** 184 **High** 185

continued from p. 96

Not only did the two leading RISCs come from academic roots but they were introduced to the commercial marketplace by systems companies rather than semiconductor manufacturers. Sun Microsystems and MIPS have taken different approaches in dealing with their semiconductor partners, but in both cases the systems company provides the CPU architecture to several IC manufacturers.

The availability of freely licensed microprocessor architectures allowed many smaller companies, such as IDT and Cypress, to enter the microprocessor business. They could do so without having to design their own architectures or develop their own software tools. As a result, the high-performance microprocessor business has become more competitive.

Into the 1990s

As the 1980s end, the war among the various processor camps is still very much underway. In the next few years, the winners and losers will emerge. This resolution will not come in 1990, however, as the next wave of RISC implementations will be a significant factor.

In 1989, the two leading CISC architectures made a leap in performance with highly integrated, pipelined implementations. With these latest, million-transistor implementations, CISC architectures essentially caught up with RISC performance levels. Unlike the i486 and 040, most of today's RISCs require external chips for cache memory, and many require external memory management or floating-point chips.

Just one decade ago, the microprocessor world was struggling with the transition from 8 to 16 bits.

In late 1990 and early 1991, companies will unveil the second generation of commercial RISCs. I expect this generation of chips to leap ahead of the i486 and the 040. Then, in 1992 and 1993, the "i586" and "68050" will do their best to catch up, and the leapfrogging may continue well into the decade.

Will the 1990s bring the same sort of changes as the 1980s? Probably not. The architectures available at the start of the 1980s had clear limitations. Today's 32-bit architectures, while certainly not ideal, are good enough to provide the basis for a few more generations of implementations. Each architecture is complete and includes a floating-point, cache, and memory-management model to guide future implementations.

Motorola and Intel evolution

Motorola started out the 1980s with the 68000—a 16-bit implementation of a fundamentally sound 32-bit architecture—and evolved it smoothly into a series of full-32-bit implementations. Unfortunately, by the time Motorola could ship the 68040, the momentum in the workstation market had shifted to RISC processors.

Intel, on the other hand, began with a flawed architecture—the segmented memory model and 20-bit address space of the 8086. Intel then proceeded to compound its mistakes in the 80286 design. The design of the 286 resulted in countless man-years wasted from force-fitting application software into Intel's segmented memory model. (A mammoth—and largely wasted—effort by Microsoft in building the OS/2 around the 286 instead of the 386 added to the problem.) Yet, in a supreme irony, the 286 forms the heart of the majority of computers on the planet. It fueled an unprecedented explosion of desktop computers. Because of the incredible power of IBM and the growth of the IBM-compatible personal computer market, Intel found itself at the center of the personal computing universe. Intel's architecture achieved this success despite the fact that this processor architecture was far from being the best.

Finally, with the 386, Intel produced a decent microprocessor architecture with a flat address space, paged memory management, and no 64K segment limitations. The incredibly lucrative monopoly Intel holds over the 386-architecture CPU market creates an inherently unstable situation, and in the early 1990s non-Intel implementations will likely appear. The 386 represents too big a market opportunity for other semiconductor companies to ignore. By the middle 1990s, the days of the proprietary instruction set will wane. The most successful architectures, at least for general-purpose computing, will come in a variety of implementations from a number of semiconductor vendors.

In late 1990 or early 1991, companies will reveal the second generation of commercial RISCs.

From architecture to implementation

A surge of RISC architectures has cropped up in the past decade to take advantage of new design ideas. The developers of the leading CISC architectures have had time to fix the most serious warts and develop high-performance implementations. The emphasis of hardware development will now shift from the introduction of new architectures to the creation of better implementations of existing architectures. It's going to be very difficult for a new architecture to gain any attention in the early part of the 1990s.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 186 **Medium** 187 **High** 188

Department

Micro View

Michael Slater
MicroDesign Resources, Inc.
(415) 494-2677
mslater@cup.portal.com

The view from 10,000 feet

The start of a new decade seems like an appropriate time to step back from the day-to-day details in which we're usually buried and take a look at where microprocessors have been in the past decade and where they will go in the next.

The blossoming of microprocessors

During the 1980s, microprocessors grew up from "toy" processors suitable only for low-performance, embedded control applications and hobbyist computers to sophisticated, powerful CPUs that challenge mainframe and supercomputer performance levels. Just one short decade ago, the microprocessor world was struggling with the transition from 8 to 16 bits. Nobody had heard of memory management, and floating-point hardware was just beginning to appear.

The most significant microprocessor event of the first half of the decade was the development of 32-bit microprocessors. They are significant not just for their wider data paths but also for their inclusion of memory management logic, floating-point support, and more capable instruction sets. The 32-bit microprocessors from Intel and Motorola established instruction-set architectures that will dominate desktop computing for the next few years, and perhaps for the rest of the century.

The emergence of RISCs

The most significant microprocessor event of the latter half of the decade was the emergence of reduced instruction-set computers as a commercial reality. Also important was the introduction of new implementations of traditional complex instruction-set architectures to challenge RISC performance. Because of the massive software base from which the Intel 80386 and i486 and the Motorola 68030/040 benefit, these CISCs have an advantage in the marketplace. This advantage prevails even though CISCs require more transistors to achieve the performance levels of RISCs. No one knows to what degree mainstream desktop computing will shift toward RISCs, but the technical workstation market is clearly making that transition.

Even though the number of RISC chips shipped so far is relatively small, the emergence of RISCs has had far-reaching effects. Traditional microprocessor designs were almost entirely products of semiconductor companies. In constrast, the early RISC designs came from academic environments.

If all new processor design had been left up to the semiconductor companies, RISC-style processors would not have resulted. Intel and Motorola had too much of a vested interest in maintaining compatibility with their existing products to consider something so radically different. Furthermore, the RISC approach was no doubt foreign to those who had worked so hard designing hardware to implement ever-more complex instruction sets. Intel and Motorola introduced RISC architectures only when it became clear that RISCs from other vendors would be a significant threat.

The development of RISCs partially resulted from the emergence of automated integrated-circuit design tools. Without such tools, microprocessor design would have remained in the hands of the semiconductor designers. Because one could design ICs without being a chip design expert or having a large staff, computer architects could create their own microprocessors.

continued on p. 95

0272-1732/90/0200/0096$1.00 © 1990 IEEE

IEEE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

Executive Committee

President: Helen M. Wood*
NOAA/NESDIS
Code E/SP, FB 4, Rm. 1069
Washington, DC 20233
(301) 763-1564

President-Elect: Duncan H. Lawrie*
Past President: Kenneth R. Anderson*

VP, Conferences and Tutorials: Laurel V. Kaleda (1st VP)*
VP, Standards: Paul L. Borrill (2nd VP)*
VP, Area Activities: Gerald L. Engel†
VP, Education: Ronald G. Hoelzeman†
VP, Membership and Information: Barry W. Johnson†
VP, Press Activities: James H. Aylor†
VP, Publications: Sallie V. Sheppard*
VP, Technical Activities: Mario R. Barbacci*

Secretary: David Pessel*
Treasurer: Joseph Boykin†
Division V Director: Edward A. Parrish, Jr.†
Division VIII Director: Roy L. Russo†
Executive Director: T. Michael Elliott†
*voting member of the Board of Governors
†nonvoting member of the Board of Governors

Board of Governors

Term Expiring 1990:
Vishwani Agrawal, Mario R. Barbacci,
Ming T. (Mike) Liu, Yale N. Patt, Donald E. Thomas,
Benjamin W. Wah, Ronald Waxman

Term Expiring 1991:
P. Bruce Berra, Michael Evangelist,
Ted Lewis, Raymond E. Miller, Earl E. Swartzlander, Jr.,
Joseph E. Urban, Thomas W. Williams

Term Expiring 1992:
Alicja I. Ellis, Tadao Ichikawa,
David Pessel, Sallie V. Sheppard, Bruce D. Shriver,
Harold Stone, Wing N. Toy

Next Board Meeting

March 2, 1990, 8:30 a.m.
Cathedral Hill Hotel, San Francisco, CA

Senior Staff

Executive Director: T. Michael Elliott
Editor and Publisher: H. True Seaborn
Director, Computer Society Press: Eugene M. Falken
Director, Conferences and Tutorials: Anne Marie Kelly
Director, Finance: Tod S. Heisler
Director, Board and Administrative Services: Violet S. Doan

Computer Society Offices

Headquarters Office
1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone (202) 371-0101
Telex: 7108250437 IEEE COMPSO
Fax: (202)728-9614

Publications Office
10662 Los Vaqueros Cir.
PO Box 3014
Los Alamitos, CA 90720-1264
Membership and General Information: (714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

European Office
13, Ave. de L'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Fax: 32 (2) 770-85-05

Asian Office
Ooshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone: 81 (3) 408-3118
Fax: 81 (3) 408-3553

Use the Reader Service Card to obtain information on:

- Membership application—student #203, others #202
- Perodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Standards working groups list #195
- Compmail+ international electronic mail/database brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures—student/regular #193
- Student scholarship information #192
- Awards description/nomination forms #198
- Volunteer leaders/staff directory #196
- IEEE senior member application #204

Purpose

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

Membership

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others seriously interested in the computer field.

Publications and Activities

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and new products.

Periodicals. The society publishes six magazines and four research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

European Office

Payments for Computer Society membership and publication orders are accepted by checks in Belgian, British, German, Swiss, or US currency. Checks in US funds must be drawn on a US bank. Payment may also be made by American Express, Eurocard, MasterCard, or Visa credit cards.

Asian Office

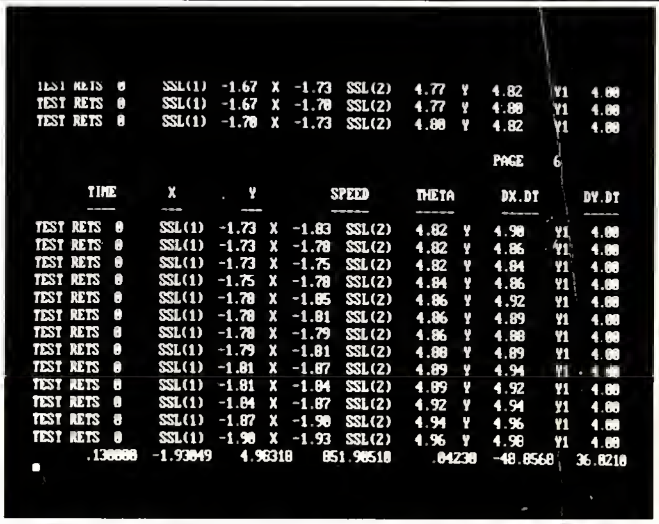
Payments for Computer Society membership and publication orders are accepted by checks in Japanese or US currency. Checks in US funds must be drawn on a US bank. Payment may also be made by electronic fund transfer to the Bank of Tokyo, Akasaka Branch, Toza acct. 0767956; the credit receiver is the IEEE Computer Society Headquarters Office. Payment may also be made by American Express, Eurocard, MasterCard, or Visa credit cards.

Ombudsman

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

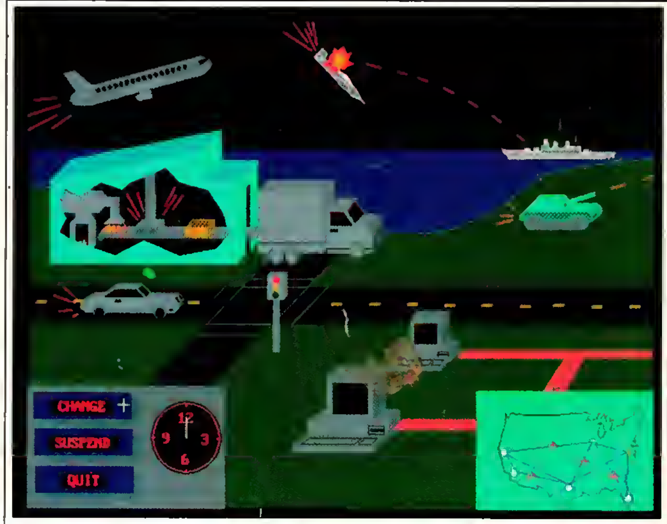
SIMULATION BREAKTHROUGH

BEFORE



Difficult to understand results

AFTER



With SIMGRAPHICS you see your simulation in action. Draw your own icons or use one of ours.

Announcing new SIMSCRIPT II.5 with SIMGRAPHICS

Simulation models are now easier to build --results are easier to understand

Now you can provide the users of your SIMSCRIPT II.5 models with SIMGRAPHICS™ --graphical input and animation.

Results are easy to understand--animated pictures, histograms, pie charts and plots.

Because simulation results are easily understood, your recommendations are more likely to be acted upon.

Simulation simplified

SIMSCRIPT II.5 gives you a compact English-like language. Your simulation program reads like a description of the system you are studying.

Your model development, check-out, modification and enhancement are greatly simplified.

Many successful applications

SIMSCRIPT II.5® is a well established, standardized, and widely used language with proven software support.

Typical applications include: military planning, manufacturing, communications, logistics, and transportation.

Free trial offer

The free trial contains everything you need to try SIMSCRIPT II.5 on your computer.

Try the SIMSCRIPT II.5 language, the quality and timeliness of our support, the accuracy of our documentation and the facilities for error-checking**--everything you need for a successful project.**

For 26 years CACI has provided trial use of its simulation software--**no cost, no obligation.**

Act now--free training offer

For a limited time we also include free training. Call today to avoid disappointment--class size is limited.

SIMSCRIPT II.5 is available on most popular computers including PC's running under OS/2, the Mac II, and workstations.

For immediate information

Call Doug Dittrich at (619) 457-9681, or Fax (619) 457-1184. In Europe, call Nigel McNamara on (01) 528-7980, Fax (01) 528-7988.

Rush information on SIMSCRIPT II.5 with SIMGRAPHICS free trial and training

Free trial-learn the reasons for the broad and growing popularity of SIMSCRIPT II.5.

No cost, no obligation.

Limited offer-return the coupon today and we will include one free course enrollment worth $950.

☐ Send information on your Special University Offer.

Name ____________________

Organization ____________________

Address ____________________

City ________ State ________ Zip ________

Telephone ____________________

Computer ________ Operating System ________

Return to: IEEE MICRO

CACI Products Company
3344 North Torrey Pines Court
La Jolla, California 92037
Call Doug Dittrich at (619) 457-9681
Fax (619) 457-1184
In Europe:
CACI Products Division
Regent House, 89 Kingsway
London WC2B 6RH, United Kingdom
Call Nigel McNamara on (01) 528-7980
Fax (01) 528-7988

SIMSCRIPT II.5, and SIMGRAPHICS are registered trademarks and service marks of CACI, INC.
©1989 CACI, INC.